Utilização de algoritmos de visão computacional para diagnóstico de doenças da vinha

Simon Afonso

Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciências de Computadores

2019

Orientador

Miguel Coimbra, Faculdade de Ciências da Universidade do Porto

Coorientador

Daniel Carvalho, UKUBO (M&D Carvalho Lda.)

Abstract

The supervision, prevention and cure of agricultural plantations diseases is key to ensuring the sustainable development of any plantation and its products. Accurate detection, and with good timing, are essential for taking correct measures of prevention and healing. The traditional methods for disease detection depend on personal experience and visual observation from the farmers.

For a person with large experience, this may not be a problem, but for an inexperienced user, can dramatically affect the resulting final product. Therefore, the need to identify plantation diseases at an early stage and suggest solutions to their owners, for damage prevention motivated us to the research and development presented in this work.

The project reported on this thesis consists on a system capable of diagnose a certain disease present on a vine leaf with the help of computer vision techniques. The main idea of this system is to extract the features of a specific disease, so we can identify it correctly when we see it the next time. We present a system where the user with an image of a vine leaf with specified capture conditions obtains a diagnosis with the identified disease, or in case of a negative diagnose, with information that the leaf is healthy.

We used a *dataset* with images of healthy leaves, as well as images of leaves infected with black rot and esca. In this dissertation we explained the procedures to fulfill the above objectives. We explained how to denoise the images from the *dataset*, how to use the *thresholding* technique to perform segmentation, which we validate against another segmentation done manually. We also talk about the different type of descriptors implemented with the goal of extract *features*, each benefit to make a final diagnostic and also the results obtained with the Support Vector Machine classifier on those *features*.

We will also explain the details of the implementation of the system, including its organization, the problems encountered during its development, and also the way we solved them.

Resumo

A supervisão, prevenção e cura de doenças em plantações agrícolas são a chave para garantir o desenvolvimento sustentável de qualquer plantação e do seu respectivo produto. A deteção precisa, e atempada destas doenças é essencial para proceder correctamente na altura de implementar as medidas de prevenção e cura. Os métodos tradicionais de deteção de doenças dependem da experiência pessoal e da observação visual dos agricultores. Para uma pessoa com vasta experiência isto pode não ser um problema, mas para um utilizador inexperiente pode alterar o seu produto final de forma drástica. Portanto, a necessidade de identificar as doenças em plantações, na sua fase inicial e sugerir soluções aos seus proprietários para prevenção de danos, motivou-nos para a busca e implementação apresentadas neste trabalho.

O projeto descrito nesta dissertação consiste num sistema capaz de diagnosticar doenças em folhas de videira através de técnicas de visão computacional. A ideia principal deste sistema é extrair características específicas de uma certa doença, para posteriormente a conseguirmos identificar quando nos encontremos novamente com esta. É apresentado um sistema de visão computacional, onde o utilizador através de uma imagem de uma folha de videira, com condições de captura específicas obtém um diagnóstico sobre o estado da folha, com a doença identificada, ou no caso de diagnóstico negativo, a informação que a folha é saudável.

Foi usado um *dataset* com imagens de folhas saudáveis, assim como de folhas infectadas com podridão negra e esca. Nesta dissertação é detalhada a forma como procedemos para cumprir os objectivos acima referidos. Explicamos como procedemos na hora de retirar ruído das imagens do *dataset*, de como utilizamos a técnica de *thresholding* para realizar a segmentação, que posteriormente validamos comparando com uma outra segmentação, realizada manualmente. Falamos ainda dos vários tipos de descritores implementados para obter as *features*, e das suas respectivas eficácias na altura de fazer um diagnóstico, e ainda dos resultados que o classificador Support Vector Machine tem sobre as características.

É ainda detalhada a implementação do sistema em si, incluíndo a forma como é organizado, os contratempos encontrados durante o seu desenvolvimento bem como a forma como foram ultrapassados.

Agradecimentos

Durante esta fase da minha vida, foram surgindo dificuldades e contratempos, mas tive a sorte de poder contar sempre com o apoio de pessoas que sempre se mostraram disponíveis e estiveram a meu lado.
Começo por agradecer ao meu orientador, o Prof. Miguel Coimbra, que apesar do pouco tempo disponível, sempre tentou partilhar a sua ampla experiência e insistiu em para que eu conseguisse ultrapassar as minhas limitações, tanto a nível académico como pessoal. Um agradecimento especial também ao Daniel Carvalho e toda a equipa da M&D Carvalho Lda., que se mostraram sempre disponíveis a ajudar em tudo o que fizesse falta e essa ajuda foi essencial para a realização deste projeto.
Aos amigos que fui conhecendo ao longo destes anos, pelo ambiente de trabalho e pelos momentos de lazer. Todos estes pequenos momentos foram importantes para o sucesso desta fase de minha vida.
Quero ainda agradecer à minha família, em especial ao meu pai e aos meus irmãos, que sempre me apoiaram e lutaram para que eu conseguisse concluir esta etapa com sucesso.
Por último, mas não menos importante (muito pelo contrário), quero agradecer à minha namorada. Nada disto seria possível sem ela. O seu apoio, paciência e sermões, foram fundamentais para me permitir chegar a este ponto.

Dedico todo este trabalho à minha mãe.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Blocos de Código

Acrónimos

ANN Artificial Neural Network

BPNN Backpropagation Neural Network

DCC Departamento de Ciência de Computadores

FCUP Faculdade de Ciências da Universidade do Porto

FLANN Fast Library for Approximate Nearest Neighbors

GLCM Grey-Level Co-Occurrence Method

GMM Gaussian Mixture Model

HR Humidade Relativa

HSV Hue Saturation Value

IDE Integrated development environment

IEEE Institute of Electrical and Electronics Engineers

IVV Instituto do Vinho e da Vinha

JPEG Joint Photographic Experts Group

KNN k-Nearest Neighbors

LBP Local Binary Pattern

MP Megapíxeis

RGB Red Green Blue

ROI Region Of Interest

SDK Software development kit

SVM Support Vector Machine

T Temperatura

TD Tangential Direction

URL Uniform Resource Locator

Capítulo 1

Introdução

Neste capítulo inicíamos por apresentar as nossas motivações para a realização deste projecto. De seguida, passámos a explicar quais os objectivo que nos propomos com esta dissertação. Finalmente, apresentamos as contribuições que esta dissertação aporta para a comunidade científica e concluímos com uma breve descrição de como são organizados os capítulos da mesma.

1.1 Motivação

Em Portugal, o sector vinícola é de grande importância tanto pelo valor económico que gera como pela população que ocupa e pelo papel que desempenha quer a nível social como de conservação do meio ambiental. O nosso país possui várias regiões vinícolas. Entre as mais conhecidas internacionalmente estão o Vinho do Porto e Douro, o Vinho da Madeira, sem esquecer o Vinho Verde da região Norte tal como se vê na Figura 1.1.

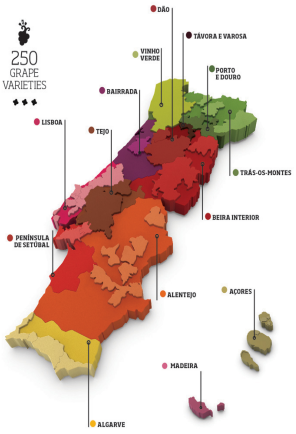

Figura 1.1: Regiões Vinícolas em Portugal
[1]

[1] Adaptado de: http://www.winesofportugal.com/us/news-and-events/news/portugal-101/

Segundo dados do Instituto do Vinho e da Vinha [1], Portugal exportou para o mundo inteiro, em 2017, quase 30 milhões de litros, o que equivaleu a quase 800 milhões de euros. A videira quando cultivada em condições climáticas favoráveis ao desenvolvimento de fungos, está sujeita a uma série de doenças, as quais poderão acarretar graves prejuízos se não forem devidamente controladas. Em Portugal, a vinha é afetada por doenças como o míldio, o oídio, a esca e a podridão negra [2].

Estas e outras doenças tem efeitos prejudiciais na vinha e levam a uma diminuição da produção, e são um dos motivos para o acontecido no ano de 2018, onde a colheita diminuiu em 3% em comparação ao ano anterior [1]. Para evitar ou amenizar estes resultados é necessário um bom tratamento da vinha e com o *timing* correcto. A principal forma de tratamento das videiras é por meio de fungicidas químicos. Deve ser prioritário o desenvolvimento de tecnologias que auxiliem na redução da aplicação destes químicos e na redução do impacto ambiental. Tecnologias que além disso melhorem a eficiência dos produtos, a escolha dos produtos certos e que evitem as condições ambientais favoráveis aos fungos. Ao diagnosticarmos de forma correcta uma doença serão aplicados os respectivos tratamentos e nas proporções adequadas. Além de levar a um aumento da probabilidade de cura da videira em si também diminui o gasto de dinheiro em químicos desnecessários e o exagero na sua utilização.

Um sistema capaz de diagnosticar a doença presente numa folha de videira através de uma imagem desta, é uma boa forma de providenciar ao utilizador uma base de decisão para quando aplicar um certo tratamento à videira. Uma ferramenta com uma alta precisão de diagnóstico, irá levar a um aumento na probabilidade de cura, pois diferentes doenças necessitam diferentes medicações. Ao acertarmos na identificação da doença iremos também reduzir a utilização de quimicos que poderão ser desnecessários para a doença em questão.

Para identificar correctamente a doença necessitámos obter as características presentes na imagem da folha em questão, e que descrevem o que está presente nesta. Uma área que possibilita esta tarefa é a visão computacional. Tal como é possível observar no Capítulo 2, visão computacional consiste essencialmente em descrever o mundo que vemos numa imagem e reconstruir as suas propriedades. Ao conseguirmos descrever as características presentes em imagens de diferentes doenças, posteriormente saberemos dizer o que necessita estar presente numa imagem para se tratar de uma doença ou outra. Como podemos ver no Capítulo 3, até ao momento, já foram realizados vários projectos baseados em visão computacional, com este mesmo objectivo. Estes projectos diferenciam uns dos outros pelas diferentes técnicas usadas para cada tarefa a realizar, seja ela pré-processamento, segmentação, a extração dos descritores em si, ou finalmente no método de classificação. Diferem ainda nos *datasets* usados em cada um, sendo que estes possuem imagens capturadas de forma diferente, e também nas doenças que estes querem diferenciar.

1.2 Objectivos

Propomos numa fase inicial, como objectivos desta dissertação, a criação de um sistema de visão computacional capaz de classificar imagens de folhas de videira consoante a doença presente nestas. Realizámos um esquema inicial, como base para o nosso projecto, que podemos observar na Figura 1.2, posteriormente, na realização do estado da arte foi confirmado como sendo o mais indicado, e sempre presente em projectos deste tipo.

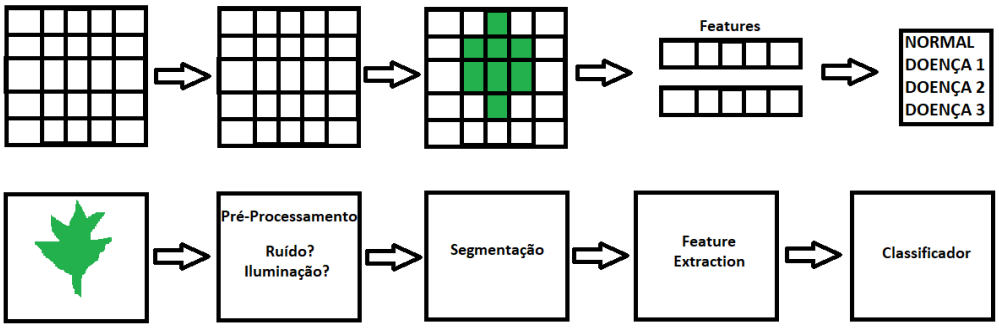

Figura 1.2: Esquema inicial do sistema

O nosso sistema deve ser capaz de eliminar, ou pelo menos, reduzir o ruído presente nas imagens do *dataset*, de modo a que os sintomas de cada doença sejam mais denunciáveis. Outro objectivo que nos propomos é a aplicação de uma segmentação próxima da real e que estabeleça da melhor forma possível as fronteiras entre os píxeis pertencentes à folha e os pertencentes ao fundo. O nosso sistema, deve também aproveitar da melhor maneira possível as características presentes em cada imagem de forma a descrever os sintomas presentes em cada uma delas. Temos como objectivo também a implementação de diferentes tipos de descritores de imagem. Finalmente, queremos que o nosso sistema seja o mais preciso possível.

1.3 Contribuições

Esta dissertação traz como contribuição principal para a comunidade científica, uma nova abordagem de utilização de técnicas de visão computacional para diagnosticar doenças em folhas de videira. Apesar de já existirem vários projectos deste tipo, como mencionado no Capítulo 3, nenhum deles faz esta conjugação de técnicas sobre este *dataset* em específico. De entre os artigos que foram analisados no capítulo mencionado, apenas um deles tinha o *dataset* disponível *online*, então optámos por usar este como *dataset* para esta dissertação. Na altura em que o artigo mencionado [3] foi desenvolvido foram usadas apenas 40 imagens de cada uma das doenças (*Leaf Blight*, Podridão Negra, Esca) e 40 imagens de folhas saudáveis. No entanto, na nossa dissertação iremos optar por usar todas as imagens do *dataset*, optando por deixar de lado as imagens de *Leaf Blight*, por esta doença não ser tão comum em Portugal. Após analisarmos de

forma exaustiva o que já foi feito, chegamos à conclusão que seria interessante investigar como é que esta conjugação de técnicas específica actua sobre este *dataset*.

1.4 Organização

A presente dissertação é composta por 6 capítulos. Este Capítulo 1, no qual nos encontrámos descreve a motivação que nos leva à realização deste projecto, os objectivos propostos assim como em que consiste realmente esta dissertação, e ainda as contribuições que esta dissertação traz para a comunidade científica.
O Capítulo 2 apresenta uma breve explicação de vários conceitos necessários para o entendimento desta dissertação. Começamos por apresentar informação sobre a planta que é a videira e as doenças que a afectam e que queremos tentar diagnosticar, assim como as características dos sintomas que estas doenças deixam nas folhas. Falamos ainda das técnicas necessárias para a realização de cada uma das etapas deste sistema. Finalizámos o capítulo com uma pequena discussão sobre as ideias que obtivemos na realização do mesmo, assim como as conclusões a que chegámos ao investigar sobre as várias componentes e técnicas do sistema.
O Capítulo 3 mostra os informação sobre os projectos com características semelhantes ao que nós realizamos. Apresentamos ainda a forma como efectuámos a pesquisa dos artigos que descrevem os ditos trabalhos, bem como a filtragem realizada à pesquisa inicial dos artigos. Finalmente falamos sobre a informação retirada destes artigos e qual desta informação é relevante para o nosso projecto.
O Capítulo 4, está dividido em três partes. As duas partes principais são o "Desenho"e o "Desenvolvimento". No primeiro apresentamos o *dataset* sobre o qual efectuamos a nossa investigação, e ainda as ferramentas usadas para a dita investigação e o motivo de escolha dessas ferramentas. Na parte de "Desenvolvimento"explicamos as acções que tomámos para obter o nosso resultado final, juntamente com a forma como todas as etapas estão interligadas. Associado a cada parte do trabalho temos imagens que demonstram como cada acção actua sobre as imagens e o que retira destas. Descrevemos ainda os problemas encontrado nestas várias etapas, e como foram resolvidos. Na última parte deste capítulo, discutimos sobre as conclusões que retirámos da implementação do sistema.
O Capítulo 5, mostra inicialmente, o nível de eficácia da nossa segmentação em comparação com uma segmentação realizada manualmente às mesmas imagens. Numa outra parte apresentamos uma análise estatística da precisão de diagnóstico dos vários descritores de imagem implementados, e debruçámo-nos sobre quais destes descritores são os melhores para diferenciar as várias doenças. Terminámos o capítulo fazendo um apanhado das conclusões que tirámos da análise realizada ao sistema.
Finalmente, no Capítulo 6 exprimimos as nossas conclusões sobre os resultados da implementação deste projecto, assim como as aptidões que esta dissertação nos trouxe. Por fim falámos sobre trabalho que poderá ser realizado futuramente, tendo este projecto como base.

Capítulo 2

Background

Neste capítulo é feito um resumo de alguns conceitos necessários para o entendimento da dissertação. São ainda apresentadas algumas doenças que afectam este tipo de plantas, os seus sintomas nas folhas, alguns conceitos de visão computacional, e várias técnicas usadas nas diferentes partes do sistema implementado. Finalmente discutimos as conclusões tiradas deste capítulo e que usámos no resto da dissertação.

2.1 Videira

Na família *Vitaceae*, é o gênero *Vitis* que tem maior importância agronômica. Consiste em aproximadamente 60 espécies que existem quase exclusivamente no Hemisfério Norte. Entre estas espécies a Vitis vinifera, mais conhecida como videira ou vinha é a única espécie amplamente utilizada na indústria do vinho a nível global [4]. A videira é uma trepadeira que possui um tronco retorcido, ramos flexíveis, folhas grandes, flores esverdeadas em ramos que posteriormente se transformam no fruto que é a uva.

Figura 2.1: Videira (*Vitis Vinifera*)
[1]

[1]Diponível em: https://www.northbaybusinessjournal.com/csp/mediapool/sites/dt.common.streams.StreamServer.cls?STREAMOID=KvZovCt48B7TY$_5$*P*7*I*80*Lc*$*daE*2*N*3*K*4*ZzOUsqbU*5*sYuQ*2*TDTgL*5*Vdq*0*SWd*6*HCFyYWCsjLu*883*Ygn*4*B*49*Lvm*9*bPe*2*QeMKQdVeZmXF*9*l*4*uCZ*8*QDXhaHEp*3*rvzXRJFdy*0*KqPHLoMevcTLo*3*h*8*xh*70*Y*6*N*$_{UC}$*ryOsw*6*FTOdKL*$_j$*pQ* − *CONTENTTYPE* = *image/jpeg*

A uva é uma das principais culturas hortícolas a nível mundial, e conta com 8 milhões de hectares de vinha em todo o mundo. É principalmente processada em vinho, mas algumas são destinadas a consumo fresco como uvas de mesa, uvas passas e ainda processadas em sumo não-alcoólico [5].

Figura 2.2: Uvas brancas (à esquerda) e tintas (à direita)
[2]

Dentro da *Vitis Vinifera* estão todas as castas de uvas que podemos identificar nos rótulos das bebidas (Figura 2.3). Explicando de maneira simples, as castas são os diferentes perfis dos frutos e videiras que existem. Ou seja, são as variedades de uvas e seus usos. Elas podem ser para o consumo da própria fruta ou para a produção de bebidas, como dito anteriormente. De entre as castas mais conhecidas temos as *Cabernet Franc*, *Merlot*, *Malbec* e o *Sauvignon Blanc*, que na totalidade soma cerca de 10 mil variações dentro da mesma espécie.

Figura 2.3: Rótulo da casta Alvarinho da sub-região de Melgaço e Monção
[3]

[2]Adaptada de: https://img.plantis.info/wp-content/uploads/2017/08/Vitis-vinifera-ssp-vinifera-2.jpg e https://www.omcseeds.com/image/cache/data/products/riverbank-grape-vitis-riparia$_4$ − 600x600.*jpg*

[3]Disponível em: http://cacho.pt/wp-content/uploads/2015/07/Soalheiro-511.jpg

2.1.1 Doenças

São várias as doenças e pragas que afectam a cultura da vinha, como são o case da esca e da podridão negra. Estas doenças apresentam sintomas ao nível das folhas que permitem a sua identificação.

Podridão Negra

Em Portugal a podridão negra, tem vindo a aumentar de incidência nalgumas regiões do País, nomeadamente nas regiões do Dão, Bairrada e Alentejo. Também conhecida como *Blackrot*. Esta doença ataca todos os órgãos da videira em fase de crescimento ativo originando estragos não só nas folhas, sendo que os maiores prejuízos resultam do ataque aos cachos levando a problemas não só de quantidade mas, também, de qualidade dos vinhos. Os primeiros sintomas surgem após um período de incubação de 3 a 4 semanas, dependendo do número de horas de humectação e temperatura. Apresenta nas folhas manchas castanho-avermelhadas, circulares ou poligonais, com uma linha marginal de cor escura e após 3 a 4 dias aparecem manchas negras, dispostas de forma concêntrica. doença é favorecida por temperatura e humidade relativa elevada (T óptima 27ºC; HR 90%), ou seja Primavera e início de Verão chuvosos.
O viticultor deve vigiar a vinha de modo a detectar o início dos primeiros focos de infecção (folhas e pâmpanos), tendo presente que os ataques sobre estes órgãos são a origem do inoculo responsável pela infecção dos cachos. Não se deve, por isso, menosprezar a protecção das infecções primárias [6].

Figura 2.4: Folhas infectadas com Podridão Negra
[4]

O tratamento pode ser através de medidas preventivas ou luta química. A primeira pode ser efectuada ao destruir as fontes de inóculo. Durante a poda, recolher e queimar varas, cachos secos, bagos afetados caídos no solo. A luta química pode ser feita através do tratamento cuidadoso dos focos primários da doença. Procurar as primeiras manchas nas folhas, recolher as folhas com manchas, e tratar a zona afetada cuidadosamente.

[4]Disponível em: PlantVillage Dataset

Esca

Esta doença leva a uma dificuldade de circulação, que se acentua nos períodos mais secos e depois da floração. Pode ser causada por grandes feridas de poda, precipitação elevada, temperaturas amenas e vento. As folhas apresentam manchas amareladas ou avermelhadas, consoante a casta seja branca ou tinta, que formam digitações entre as nervuras e depois secam [2].

Figura 2.5: Folhas infectadas com Esca
[5]

As medidas a adoptar no tratamento desta doença são de carácter preventivo, visando limitar as contaminações e as fontes de inóculo. Deve-se inicialmente proceder a uma limpeza cuidada dos terrenos antes da implantação de uma nova vinha e utilizar material são na implantação de vinhas novas. Tendo em conta que há descontinuidade na manifestação dos sintomas apresentados pelas videiras com esca, as vinhas devem ser observadas durante 3 a 4 anos consecutivos, marcando-se, em cada ano, as videiras que mostrarem sintomas da doença. Deve-se ainda cortar e queimar as videiras mortas, bem como os sarmentos que apresentem sintomas (manchas de coloração castanha clara a escura, e de consistência dura, podendo também a madeira estar esfarelada). Como medida de prevenção final, deve-se podar o mais tarde que for possível. Como tratamento químico, é recomendável o uso de produtos à base de cobre [7].

[5]Adaptado de: http://www.agrotec.pt/userfiles/image/dropzone/blogs/20180718004810-vinha.jpg e https://www.cm-oaz.pt/imagem/Esca.bmp e

2.2 Visão Computacional

Segundo Szeliski [8], visão computacional consiste essencialmente em descrever o mundo que vemos numa ou mais imagens e reconstruir as suas propriedades, tais como a forma, a iluminação e a distribuição de cores. O ser humano e os animais conseguem fazer esta tarefa sem esforço, no entanto os algoritmos de visão computacional são propensos a erros. A visão computacional em si existe desde a década de 1960, mas só recentemente foi possível construir sistemas computacionais úteis usando ideias deste tipo. Este florescimento tem sido impulsionado por vários motivos, tais como, os computadores e sistemas de imagem serem muito mais baratos atualmente, além da nossa própria compreensão da geometria básica e física subjacente à visão e, mais importante, o que fazer a com ela, também ter melhorado significativamente ao longo dos anos. Isto implica que seja mais fácil fazer pesquisas sobre o assunto em questão. Este conceito de visão computacional está a ser usado atualmente numa ampla variedade de aplicações do mundo real. A maioria destas aplicações têm os componentes em comum apresentados na Figura 2.6 que são essenciais para o seu funcionamento.

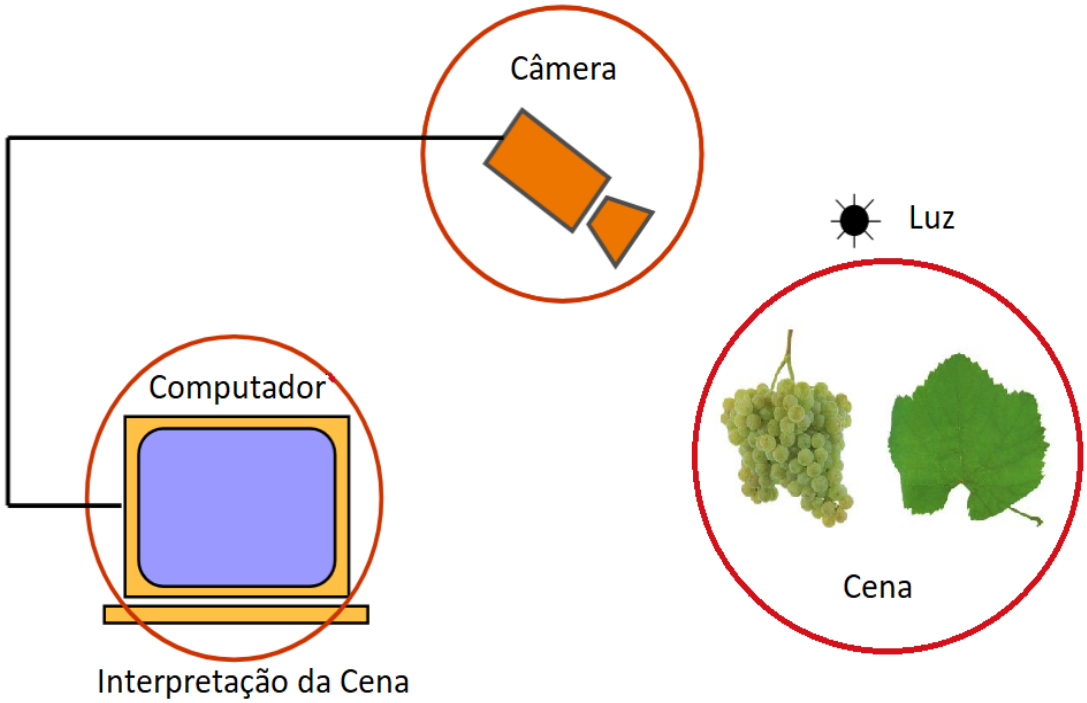

Figura 2.6: Componentes necessários num sistema de visão computacional
[6]

[6] Adaptado de: https://www.dcc.fc.up.pt/ mcoimbra/lectures/VC$_1$819/$VC_1819_TP2_ImageFormation.pdf ehttp$: $//www.winesofportugal.info/images_wop/seccao_patrimonio/castas2/brancas/alvarinho.jpg$

É possível construir sistemas de software que melhoram imagens e identificam fenómenos ou eventos importantes a acontecer na imagem a ser analisada [9]. Além de outras utilidades, este tipo de sistema é útil para análise e classificação de doenças em folhas de videiras. Doenças com características visíveis a nível das folhas são possíveis de identificar com um sistema como o da Figura 2.7, com técnicas apropriadamente associadas a cada parte deste mesmo sistema.

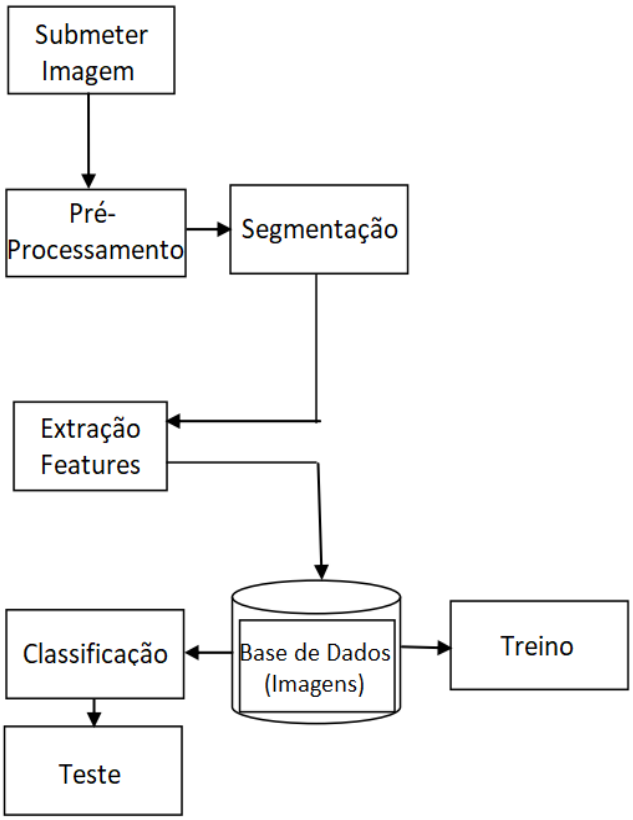

Figura 2.7: Sistema de Classificação de Imagens
[7]

2.2.1 Pré-processamento

Segundo Gonzalez e Woods [10] o principal objetivo do pré-processamento é modificar uma imagem para que o resultado final seja mais adequado do que a imagem original para uma aplicação específica. Lida principalmente com a melhoria dos dados da imagem, ao suprimir ruídos indesejados e ao aprimorar algumas características importantes da imagem em questão. Este passo é útil para as futuras tarefas de processamento e análise. Uma forma de efectuar pré-processamento é a implementação de técnicas de regulação de iluminação, como é o caso do *contrast stretching*. Estas técnicas fazem com que as características da imagem sejam mais facilmente detectáveis.

[7] Adaptado de: [3]

Regularização da Iluminação

Técnicas deste tipo, tem como objectivo compensar os efeitos das variações da iluminação, corrigir sombras, preservando ao mesmo tempo os elementos essenciais da aparência visual. A iluminação ou brilho refere-se à luminosidade geral ou escuridão da imagem enquanto que o contraste é a diferença de brilho entre objetos ou regiões. Se a técnica de *contrast stretching* vai actuar sobre o contraste, como iremos ver a seguir, esta irá trabalhar sobre a luminosidade. A Figura 2.8 mostra quatro desajustes possíveis de brilho e de contraste, onde podemos assimilar a diferença entre estes dois conceitos. Quando o brilho é muito alto, como em (a), os pixéis mais brancos estão saturados, destruindo os detalhes nessas áreas. O contrário acontece em (b), onde o brilho é muito baixo, saturando os pixéis mais escuros. A figura (c) mostra o contraste definido como alto, resultando em pretos muito pretos e brancos muito brancos. Por último, (d) tem o contraste definido muito baixo; Todos os pixels são num tom médio de cinza, fazendo os objetos desvanecer uns dos outro [11]. A utilidade da regularização da iluminação reside em aumentar o brilho para eliminar o ruído das sombras ou reduzir o brilho para eliminar o ruído dos reflexos na captura da imagem, tornando mais fácil, tal como na técnica anterior, a identificação das suas características.

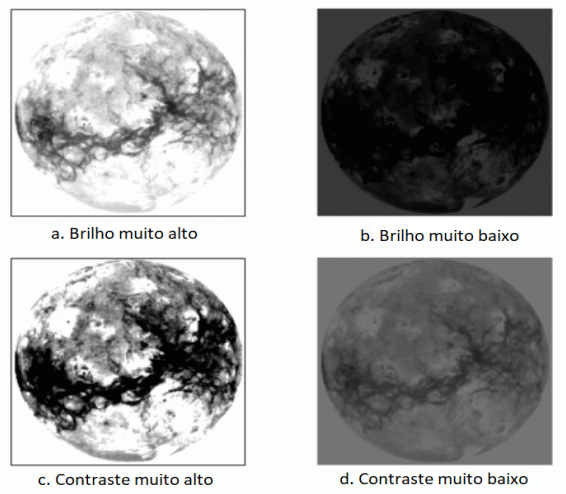

Figura 2.8: Diferentes desajustes de brilho ou contraste
[8]

Contrast Stretching

Contrast Stretching (também conhecido como Normalização) é frequentemente usado para aprimorar imagens de baixo contraste, que podem resultar de iluminação deficiente, falta de *dynamic range* (razão entre os valores mais alto e mais baixo) no sensor de imagem, configuração incorreta da abertura da lente durante a aquisição da imagem, entre outros [12]. A técnica de *Contrast Stretching* tenta melhorar uma imagem esticando a faixa de valores de intensidade que ela contém para aproveitar ao máximo os valores possíveis, como apresentado na Figura 2.9. Esta técnica é restrita a um mapeamento linear de valores de entrada para saída.

[8] Adaptado de: https://www.dspguide.com/graphics/F_23_10.gif

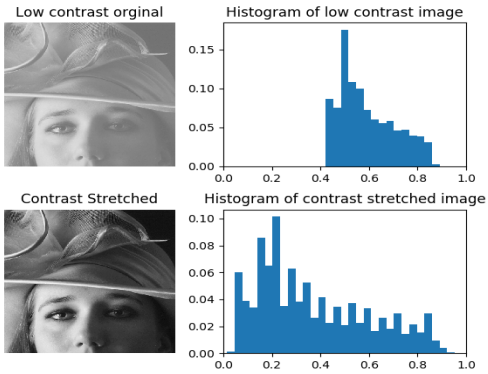

Figura 2.9: Exemplo de Contrast Stretching
[9]

O primeiro passo do *contrast stretching* é determinar os limites sobre os quais os valores de intensidade de imagem são estendidos. Esses limites inferior e superior são chamados a e b, respectivamente (para imagens padrão em escala de cinza de 8 bits, esses limites são geralmente 0 e 255). Em seguida, o histograma da imagem original é examinado para determinar os limites de valor (inferior = c, superior = d) na imagem não modificada. Se o intervalo original cobrir o conjunto de valores possíveis, o alongamento simples do contraste não conseguirá nada, mas mesmo assim, como às vezes, a maioria dos dados da imagem está contida num intervalo restrito, esse intervalo restrito pode ser esticado linearmente, com valores originais que estão fora do intervalo definido para o limite apropriado do intervalo de saída estendido. Então, para cada pixel, o valor original r é mapeado para o valor de saída s usando a função:

$$s = (r - c)(\frac{b - a}{d - c}) + a$$

Um problema com este método é que os *outliers* podem reduzir a eficácia da operação. Frequentemente é vantajoso selecionar c e d que estejam, respectivamente, nos percentis 5 e 95 dos valores de entrada. Alternativamente, pode-se começar no pico do histograma e subir e descer na lista de valores até que apenas um pequeno número (por exemplo, 1% a 3%) de valores seja rejeitado e deixado fora dos limites escolhidos para c e d.

2.2.2 Segmentação

Novamente, segundo Gonzalez e Woods [10], a segmentação é utilizada para sub-dividir uma imagem em diferentes áreas, regiões ou objectos. O nível ao qual esta sub-divisão deve ser levada depende do problema em questão. A segmentação de imagens não triviais é uma das tarefas mais difíceis no processamento de imagens. A precisão da segmentação determina o eventual sucesso ou fracasso dos próximos passos do sistema. Uma das técnicas mais usadas para segmentação é o *thresholding*.

[9] Adaptado de: https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/_images/contraststretch.png

Thresholding

Arora et al. [13] explicam que *thresholding* é uma técnica importante de segmentação de imagens. O objetivo de uma segmentação eficaz é separar objetos do fundo e diferenciar píxeis com valores próximos para melhorar o contraste. Em muitas aplicações de processamento de imagem, espera-se que os píxeis pertencentes à mesma região tenham características homogêneas (por exemplo, nível de cinza ou cor), que indicam que pertencem ao mesmo objecto. As técnicas de *thresholding* podem ser divididas em *bi-level* e *multi-level*, dependendo do número de segmentos da imagem. Com *bi-level thresholding*, a imagem é segmentada em duas regiões diferentes. Os píxeis com valores de cinza maiores que um determinado valor T são classificados como píxeis de objecto, e os outros valores de cinza menores que T são classificados como pixéis de fundo, tal como na Figura 2.10.

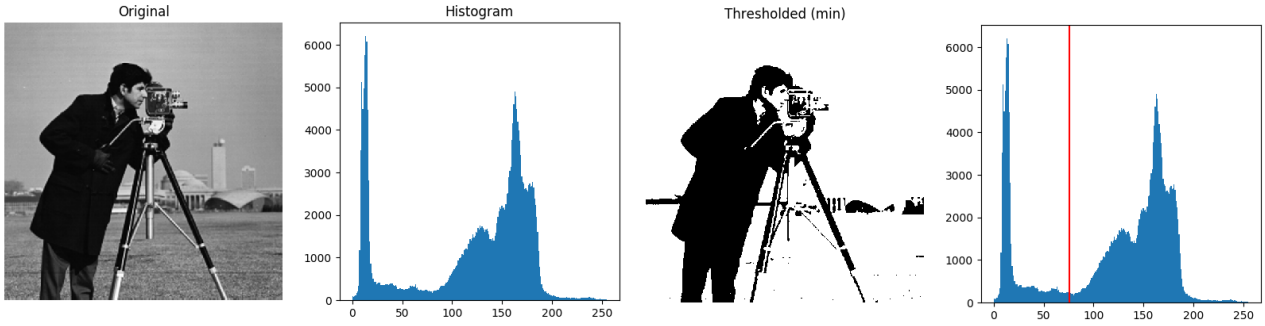

Figura 2.10: Exemplo de *bi-level thresholding*
[10]

Multi-level thresholding segmenta uma imagem em nível de cinza em várias regiões distintas. Esta técnica determina mais de um limite para a imagem determinada e segmenta a imagem em determinadas regiões de brilho, que correspondem a um segundo plano e vários objectos. Este método funciona muito bem para objectos com fundos coloridos ou complexos, nos quais o *bi-level thresholding* não produz resultados satisfatórios.

Figura 2.11: Exemplos de *multi-level thresholding* (3-level, 4-level e 5-level *thresholding*)
[11]

[10] Adaptado de: https://scikit-image.org/docs/0.14.x/_images/sphx_glr_plot_thresholding_003.png
[11] Disponível em: http://file.scirp.org/pdf/JILSA20100300001_5_6098874.*pdf*

Filtros Morfológicos

Segundo Gonzalez e Woods [10], operadores morfológicos tais como *dilate*, *erode*, *open* e *close*, podem ser aplicado através dos filtros de imagem para aumentar e diminuir regiões da imagem, assim como para remover ou preencher píxeis nas fronteiras entre as regiões da imagem.
Dilation e *erosion* são operadores básicos na área da morfologia matemática. O efeito básico do ***dilation*** numa imagem é a ampliação gradual dos píxeis das fronteiras das regiões, geralmente píxeis brancos. À medida que a área dos píxeis do objecto em primeiro plano crescem, os buracos dentro dessas regiões diminuem. O efeito básico do ***erosion*** numa imagem é corroer os píxeis das fronteiras das regiões do objecto em primeiro plano. À medida que a área dos píxeis em primeiro plano diminuem de tamanho, os buracos dentro dessa área tornam-se maiores.
Ao fazer uma *dilation* e uma *erosion* usando como elemento estrutural um quadrado 3x3 numa imagem binária obtém-se a transformação mostrada nas Figuras 2.12 e 2.13. Se usarmos elementos estruturais maiores teremos efeitos mais extremos. Com elementos estruturais maiores, é bastante comum usar uma forma aproximadamente redonda, em vez de um quadrado.

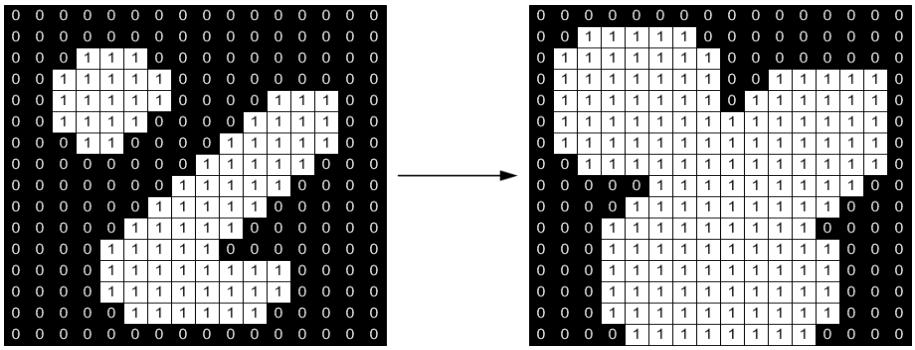

Figura 2.12: Efeito de uma operação de *dilate* numa imagem binária (com quadrado 3x3) [12]

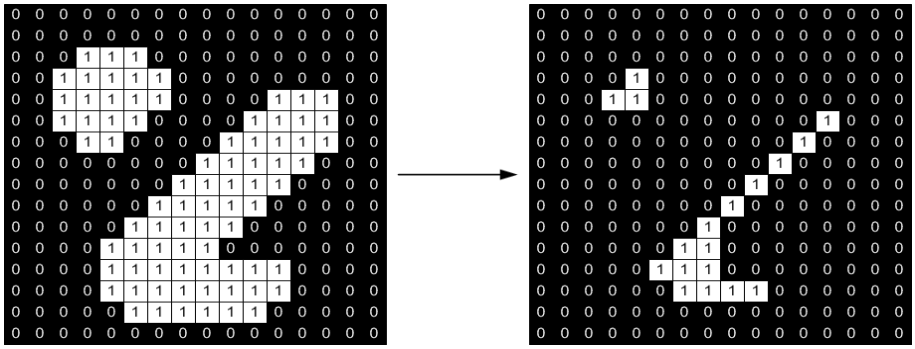

Figura 2.13: Efeito de uma operação de *erode* numa imagem binária (com quadrado 3x3) [13]

[12] Disponível em: https://www.theobjects.com/dragonfly/dfhelp/30/Content/Resources/Images/Image%20Processing/MorphologyFilter_Dilate.png
[13] Disponível em: https://www.theobjects.com/dragonfly/dfhelp/30/Content/Resources/Images/Image%20Processing/MorphologyFilter_Erode.png

O ***opening*** é derivado das operações de *erosion* e *dilation*. O efeito básico do *opening* é semelhante ao *erosion*, pois tende a remover alguns dos píxeis de primeiro plano (claros) dos píxeis nos limites das regiões. Em geral, é menos destrutivo que o *erosion*. O efeito do operador é preservar as regiões de primeiro plano que têm uma forma semelhante ao elemento estrutural usado, ou que pode conter completamente o elemento estrutural, enquanto elimina todas as outras regiões dos píxeis de primeiro plano. Devemos ter em mente que fazer *open* dos píxeis de primeiro plano com um elemento estrutural específico é equivalente a fazer *close* dos píxeis de fundo com o mesmo elemento.

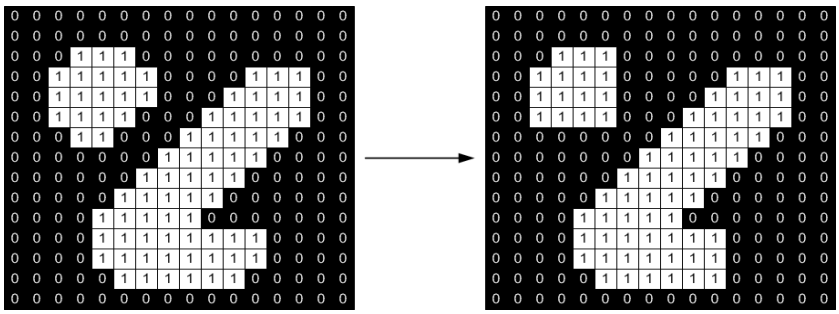

Figura 2.14: Efeito de uma operação de *open* numa imagem binária (com quadrado 3x3)
[14]

O ***closing*** pode ser definido simplesmente como uma dilatação seguida por uma erosão usando o mesmo elemento estrutural para ambas as operações. O *close* é similar em alguns aspectos ao *dilate*, pois tende a ampliar os píxeis dos limites das regiões de primeiro plano (brilhantes) e preenche pequenos orifícios de fundo conhecidos como *pepper noise*. Como com outros operadores morfológicos, a operação exata é determinada pelo elemento estrutural usado. Tem como objectivo preservar as regiões de fundo que possuem uma forma semelhante a esse elemento, ou que pode conter completamente o elemento, enquanto elimina todas as outras regiões de píxeis de fundo.

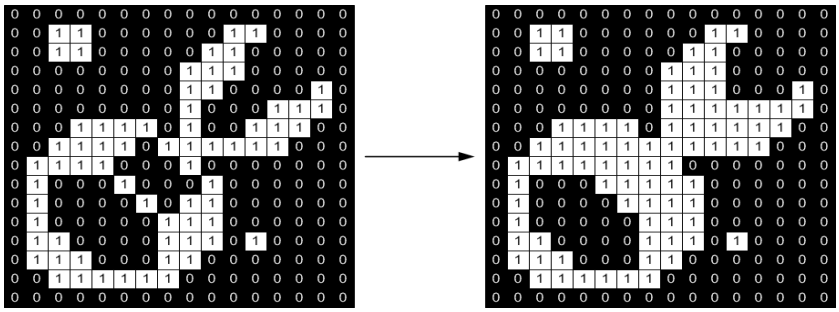

Figura 2.15: Efeito de uma operação de *close* numa imagem binária (com quadrado 3x3)
[15]

[14]Disponível em: https://www.theobjects.com/dragonfly/dfhelp/30/Content/Resources/Images/Image%20Processing/MorphologyFilter_Open.png

[15]Disponível em: https://www.theobjects.com/dragonfly/dfhelp/30/Content/Resources/Images/Image%20Processing/MorphologyFilter_Close.png

2.2.3 *Feature Extraction*

Segundo Kumar e Bhatia [14], *feature extraction* descreve as informações relevantes contidas numa imagem para que a tarefa de classificar a imagem seja facilitada por um procedimento formal. No reconhecimento de padrões e no processamento de imagens, o conceito *feature extraction* é uma forma especial de redução na dimensão dos dados. O principal objectivo de *feature extraction* é obter as informações mais relevantes dos dados originais e representar essas informações num espaço de menor dimensão. Quando os dados de entrada para um algoritmo são muito grandes para serem processados e alguns destes dados são suspeitos de ser redundantes (muitos dados, mas não muita informação), os dados de entrada são transformados num conjunto de *features* de representação reduzida (também denominado vector de *features*).
Se as *features* extraídas forem cuidadosamente escolhidas, espera-se que o conjunto de *features* consiga extrair informações relevantes dos dados de entrada para executar a tarefa desejada com o uso dessa representação reduzida em vez da entrada inicial (imagem completa) com tamanho muito maior. A seleção de *features* é essencial para todo o processo do sistema, pois os futuros passos do sistema como o classificador não serão capazes de reconhecer *features* mal selecionados. O passo de *feature extraction* é crucial na construção de qualquer classificador de padrões e visa a extracção de informações relevantes que caracterizam cada classe. Os tipos de *features* possíveis de extrair de uma imagem são de cor, textura e de forma.

Cor

Uma forma de representar as *features* de cor é através de um histograma. Este histograma consiste numa representação do número de píxeis com determinado valor do campo do espaço de cores em questão. Dois espaços de cores possíveis de utilizar são RGB (*Red Green Blue*) e HSV (*Hue Saturation Value*). Cada campo do respectivo espaço de cor representa uma característica específica que salienta as informações possíveis de retirar de uma imagem de diferentes formas, tal como mostra a Figura 2.16.

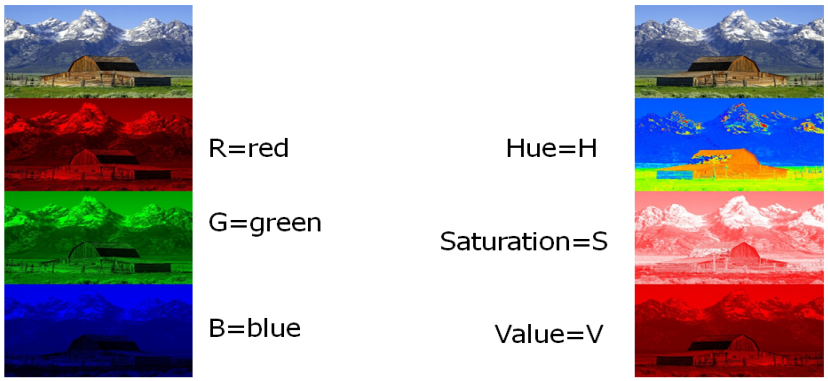

Figura 2.16: RGB vs HSV
[16]

[16]Disponível em: https://guiguilegui.files.wordpress.com/2018/02/rgb_hsv.png

Uma representação tridimensional do **espaço de cores RGB** é um cubo tal como o da Figura 2.17. Como podemos ver na figura, a posição (255,255,255) representa a cor branca e o contrário acontece na posição (0,0,0) onde se encontra a cor preta. Ao longo de cada um dos eixos encontrámos diferentes tonalidades da respectiva cor associada ao eixo. É possível então retirar *features* de uma imagem tendo em conta os valores de R, G, e B para cada píxel da imagem.

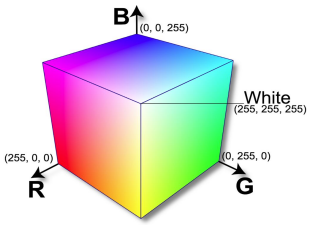

Figura 2.17: Espaço de cor RGB
17

Por outro lado, Sural et al. [15] explicam que, uma representação tridimensional do **espaço de cor HSV** é um cone hexagonal, onde o eixo vertical central representa a Intensidade (*Value*). O Tom (*Hue*) é definido como um ângulo que varia entre 0º e 360º em relação ao eixo Vermelho. O vermelho aparece no ângulo 0, verde em 120º, azul em 240º e vermelho novamente em 360º. A Saturação (*Saturation*) é a profundidade ou pureza da cor e é medida como uma distância radial do eixo central com valor entre 0 no centro e 1 na superfície externa. Para S = 0, à medida que se move para cima ao longo do eixo *Value*, passa-se de preto para branco através de vários tons de cinza.

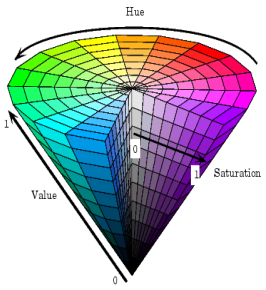

Figura 2.18: Espaço de cor HSV
18

Considerando uma determinada intensidade e tom, se a saturação for alterada de 0 para 1, a cor muda de um tom de cinza para a forma mais pura da cor representada pelo seu tom. Visto de outra forma, qualquer cor no espaço do HSV pode ser transformada num tom de cinza, baixando suficientemente a saturação. O valor da intensidade determina a tonalidade cinza específica à qual essa transformação converge. A saturação dá uma ideia sobre a profundidade da cor, o olho humano é menos sensível à sua variação comparado à variação no tom ou na intensidade.

[17] Disponível em: https://cdn.imgbin.com/7/20/16/imgbin-rgb-color-space-rgb-color-model-light-light-R3MLVE6DN37EnGLJHY0MUqtS9.jpg

[18] Disponível em: https://pdfs.semanticscholar.org/cbae/0cf18596ea501962ac93d77d50fe2ff91e6d.pdf

Textura

A textura de uma imagem fornece-nos informações sobre o arranjo espacial da cor ou intensidades de uma imagem. Uma técnica para obter a textura de uma imagem é ***Local Binary Pattern* (LBP)**, que segundo Heikkilä et al. [16], possui propriedades que favorecem o seu uso na descrição da região de interesse, como tolerância a alterações de iluminação e simplicidade computacional. O histograma do *local binary pattern* calculado sobre uma região é usado para a descrição da textura. Esta técnica descreve cada píxel pelo nível de cinza relativamente aos seus píxeis vizinhos, como no exemplo da Figura 2.19.

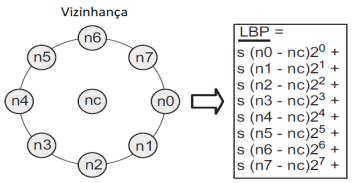

Figura 2.19: LBP *features* para uma vizinhança de 8 pixéis
[19]

Se o nível do grafo do pixel vizinho for maior ou igual, o valor é definido como 1, caso contrário, para 0. O descritor descreve o resultado sobre a vizinhança como um número binário:

$$LBP_{R,N}(x,y) = \sum_{i=0}^{N-1} s(n_i - n_c)2^i, \quad s(x) = \begin{cases} 1, & x \geqslant 0, \\ 0 & \text{otherwise,} \end{cases}$$

onde nc corresponde ao nível de cinza do píxel central de uma vizinhança local e ni aos níveis de cinza de n pixéis igualmente espaçados num círculo de raio R. Como a correlação entre píxeis diminui com a distância, muitas informações de textura podem ser obtidas de vizinhanças locais. Assim, o raio R é geralmente pequeno. Na prática, a equação acima significa que os sinais das diferenças numa vizinhança são interpretados como um número binário de N bits, resultando em valores distintos de $2N$ para o padrão binário. A informação sobre a textura de uma imagem, pode mais uma vez ser representada através de um histograma com o número de píxeis com nível de cinza correspondente a um certo intervalo, tal como no *LBP Histogram* da Figura 2.20.

Figura 2.20: Exemplo de aplicação de LBP
[20]

[19] Adaptado de: https://www.semanticscholar.org/paper/Combining-weighted-adaptive-CS-LBP-and-local-linear-Zhang-Zhang/acba2c02c9be8f6afe1e54a2ede300d564d46aa0/figure/0

[20] Disponível em: https://www.semanticscholar.org/paper/Facial-Based-Attendance-Monitoring-System-Using-Tamboli-Sardeshmukh/f3163ef817c985c25694b5264df40bfe0e4d3539/figure/3

Forma

Segundo Patel e Tandel [17], a forma é uma característica importante que descreve o conteúdo de uma imagem. Mas devido ao ruído e oclusão, entre outros factores, a forma da imagem é frequentemente corrompida e o problema de reconhecimento de objectos torna-se mais complexo. A representação da forma baseia-se principalmente nas características da forma, que são baseadas na informação dos seus limite ou dos limite mais o conteúdo interior. Uma das características importantes de um descritor de forma é a baixa complexidade computacional. Ao envolver menos propriedades da imagem no processo de computação, o cálculo pode ser minimizado e uma menor complexidade de computação alcançada, desta forma o descritor de forma torna-se robusto.

Com base em se as *features* de forma são extraídas apenas do contorno ou são extraídas da região da forma completa, as técnicas de representação e descrição da forma são classificadas em duas categorias: Região e Contorno.

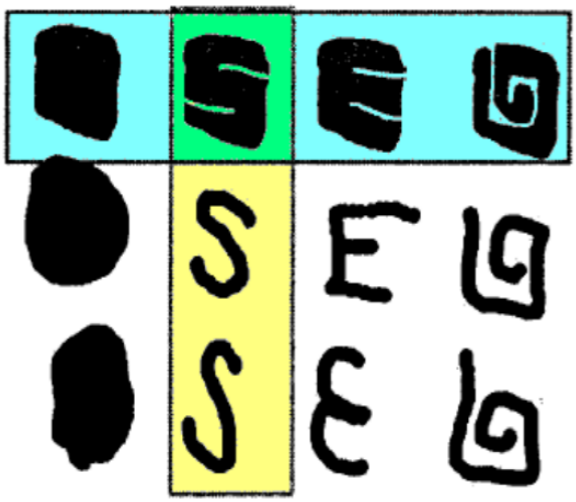

Figura 2.21: Semelhanças baseadas em regiões (azul) e em contornos (amarelo)
[21]

Quando usámos técnicas de **contorno**, é extraída informação sobre os limites da imagem, por outro lado, nas técnicas baseadas em **regiões**, todos os pixéis dentro da região da forma, isto é, a região inteira é levada em consideração para a representação e descrição da forma.

[21] Disponível em: https://slideplayer.com/slide/2801364/ (Slide 37)

2.2.4 Classificação

Classificação em *machine learning* é o processo de prever a classe a que pertencem determinados dados. Digamos que temos uma loja e queremos descobrir se um dos nossos clientes visitará a nossa loja novamente ou não. A resposta a essa pergunta pode ser "Sim" ou "Não", estas duas respostas são consideradas classes. Os problemas de classificação normalmente têm uma saída categórica como "sim" ou "não", "1" ou "0", "Verdadeiro" ou "Falso". Como no exemplo anterior, imaginemos que temos uma loja, e queremos verificar se um dos nossos clientes vai voltar à loja. Para isso, as condições da loja e do atendimento são factores dependentes, e a resposta do cliente vai ser baseada nestes factores. Baseando-nos em experiência anterior (treino) temos que considerar que factores tem em comum os clientes que voltaram à loja e quais os que tem em comum os que não voltaram. Assim quando quisermos descobrir se um novo cliente irá voltar só temos que ver a quais factores o cliente dá importância e comparar com o que retirámos da nossa experiência
No caso da classificação de uma imagem os factores são as características retiradas desta. Esta tarefa é dividida em duas fases, nomeadamente a fase de treino e a fase de teste.

- **Fase de treino**: é criado um algoritmo que é posteriormente utilizado para classificar uma imagem baseado nas *features* extraídas anteriormente. As características retiradas das imagens treinadas são armazenadas numa base de dados.
- **Fase de teste**: são retiradas as *features* das imagens que queremos classificar e são então comparadas com as armazenadas na base de dados e será aplicado o algoritmo criado para obter a respectiva classe.

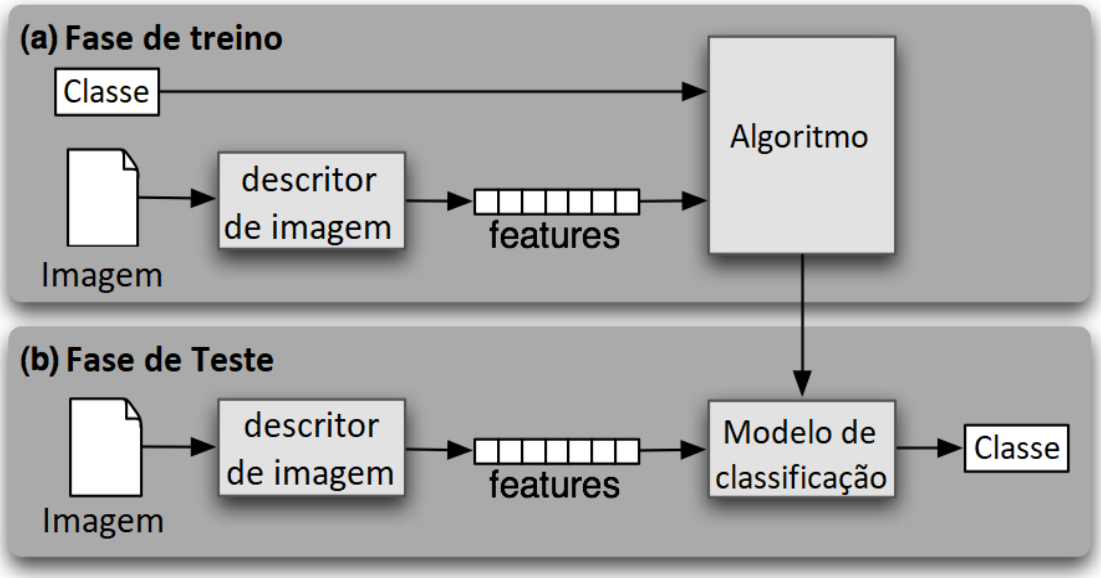

Figura 2.22: Classificação

Support Vector Machine

Um possível algoritmo para classificação são as *Support Vector Machines* (SVM). Este algoritmo é um classificador que usa um hiperplano chamado 'limite de decisão' entre duas classes. Este hiperplano tenta dividir, uma classe que contém o vector de treino rotulado como +1 de outra classe que contém o vector de treino rotulado como -1. Com o uso destes vectores de treino rotulados, o optimizador SVM encontra um hiperplano que maximiza a margem que separa as duas classes, como mostra a Figura 2.23 [18].

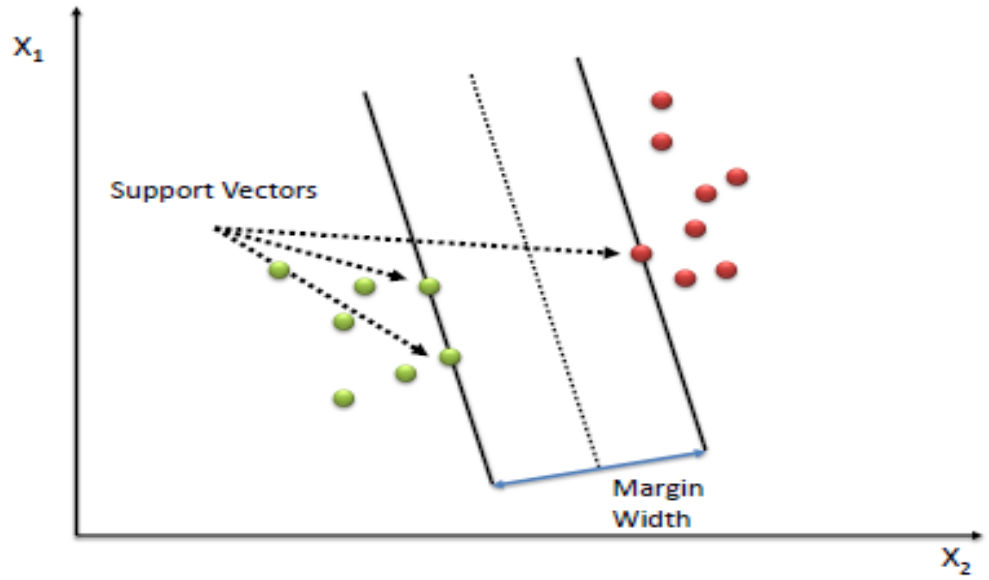

Figura 2.23: - Support Vector Machine
[23]

Explicando melhor, este método de classificação tem como princípio ajustar o limite de decisão a uma região de pontos que são todos iguais (isto é, pertencem a uma mesma classe). Uma vez que o limite é ajustado (na amostra de treino), para quaisquer novos pontos (amostra de teste) que queiramos classificar, devemos simplesmente verificar se eles estão dentro desse limite ou não. A maior vantagem das SVM é que, quando um limite é estabelecido, a maioria dos dados de treino passam a ser redundantes. Passa a ser apenas necessário um conjunto básico de pontos que podem ajudar a identificar de que lado do limite os novos pontos se encontram. Esses pontos de dados são chamados de vectores de suporte (*support vectors*) porque 'suportam' (apoiam) o limite. Num exemplo simples de duas dimensões (duas classes), este limite pode ser uma linha reta ou uma curva. Em três dimensões, pode ser um plano ou uma superfície complexa. Em dimensões mais altas são impossíveis de visualizar e 'hiperplano' é, portanto, um nome genérico para o limite em mais de 3 dimensões.
Existem vários hiperplanos a separar as classes, mas aquele que garante que a distância geométrica média entre as classes é maximizada é o melhor. Essa distância (n-dimensional) é chamada de

[22] Adaptado de: https://cdn-images-1.medium.com/max/1600/0*NNBbozJqGBpwh3M8.png
[23] Disponível em: https://i.imgur.com/EgYpim1.jpg

margem. Portanto, um algoritmo SVM essencialmente executa um esquema de optimização para maximizar essa margem. Mas nem sempre é possível garantir que os dados possam ser separados de forma clara. Pode ser raro descobrir que os dados são linearmente separáveis. Quando isto acontece, pode continuar a haver muitos pontos dentro da margem. Neste caso, o melhor hiperplano é aquele que possui um número mínimo de pontos na margem. Para garantir isso, é aplicada uma 'penalização' por cada 'contaminante' dentro da margem e o hiperplano que tem o custo mínimo de penalização total é o escolhido.

2.3 Discussão

Com a realização deste capítulo, conseguímos ter uma pequena noção de algumas técnicas usadas nesta área, e que são necessárias para o entendimento do resto da dissertação. Conseguímos também concluír que todo o processo a realizar é bastante complexo e é necessário prestar atenção aos detalhes para não induzir em erro o diagnóstico e o respectivo tratamento aplicado na videira. Ficamos a perceber que cada uma destas tarefas que foram implementadas tem um objectivo muito específico e essencial para os passos seguintes.

Capítulo 3

Estado da Arte

Neste capítulo é apresentado o estado da arte para esta dissertação. Tentámos apresentar a metodologia usada na obtenção dos artigos. Apresentámos também as especificações, técnicas e procedimentos utilizados em cada um destes artigos. Por fim, discutimos as informações importantes que extraímos deste estado da arte, aspectos comuns à maioria, e uma base sobre qual trabalhar.

3.1 Metodologia

Com o intuito de obter mais informação sobre quais são as técnicas e procedimentos mais utilizados e/ou mais adequados para a realização de um projecto como este, fizemos várias pesquisas de artigos em diferentes bases de dados. As duas escolhidas foram o *IEEE Xplore* e o *Google Scholar*. Tentamos focar na combinação de processamento de imagem com doenças da vinha e limitando os sintomas às folhas. Então decidimos que a *query* mais adequada para ser utilizada em ambas as bases de dados seria a seguinte:

(Vineyard Disease OR Grape Disease) AND Image Processing AND Leaf

Os resultados obtidos inicialmente foram:

IEEE Xplore -> 13 artigos
Google Scholar -> 135 artigos

De seguida optámos por limitar os resultados para artigos publicados depois do ano de 2012, não só com o objectivo de reduzir o número de resultados, mas também para obter artigos mais actuais. Com esta actualização os resultados foram os seguintes:

IEEE Xplore -> 12 artigos
Google Scholar -> 69 artigos

Com este filtro dá para notar que houve uma grande diminuição no número de artigos, especialmente na base de dados do Google Scholar. Como este número nos pareceu bastante razoável seguimos para a leitura dos títulos destes 81 artigos. Antes da leitura ainda eliminámos 3 artigos que estavam num idioma que não entendemos (Chinês). Dos 78 artigos resultantes da pesquisa inicial, decidimos que 24 pareciam interessantes e adequados para o tema da dissertação. A partir deste número, passamos para a leitura dos *Abstracts*, de onde eliminamos 8, por não terem as características necessárias do projecto. Por fim destes 16 que sobraram passámos para a leitura completa dos artigos e obtivemos um resultado final de 10 artigos. Todo este processo de filtragem está especificada na Figura B.1.

3.2 Resultados

Neste momento já foram muitos os projectos realizados com vista a diagnosticar doenças em folhas de várias plantas com o uso de técnicas de visão computacional. De entre os projectos que tratam exclusivamente de folhas de videira merecem especial atenção os seguintes:

Tabela 3.1: - Técnicas Usadas nos Artigos

Referência	Segmentação	Feature Extraction	Classificação	Doenças
Krithika e Selvarani [19]	Tangential Direction (TD)	Cor: HSV e L*a*b* color spaces Textura: Grey-Level Co-Occurrence Method (GLCM)	K-Nearest Neighbors (KNN)	
Hase et al. [20]	Gaussian Mixture Model (GMM)	Cor: HSV	Fast Library for Aproximate Nearest Neighbors (FLANN)	• Bactéria (93,2%) • Fungo (94,7%)
Elemmi et al. [21]		Cor, Textura e Forma	Support Vector Machine (SVM)	• Míldio | • Oídio • Antracnose | • Bactéria (96,6%)
Agrawal et al. [3]	K-means clustering	Cor: HSV e L*a*b* Textura: GLCM	Multi-class Support Vector Machine (SVM)	• Podridão Negra | • Esca • Leaf Blight (90%)
Waghmare e Kokare [22]	Multi-level Thresholding	Cor: HSV Textura: Opposite Colour Linear Binary Pattern	Multi-class Support Vector Machine (SVM)	• Míldio | • Oídio (96,6%)
Sandika et al. [23]	Region Of Interest	Textura: GLCM e Local Binary Pattern (LBP)	Random Forest	• Míldio | • Oídio • Antracnose (86%)
Padol e Yadav [18] (A) Padol e Sawant [24] (B)	K-means clustering	Cor: HSV Textura	(A) Support Vector Machine (SVM) (B) Support Vector Machine (SVM) & Artificial Neural Network (ANN)	(A) • Míldio (93,3%) | • Oídio (83,3%) (B) • Míldio (100%) | • Oídio (100%)
Kharde e Halkarai [25]	Watershed Algorithm	Textura: GLCM	Artificial Neural Network (ANN)	• Podridão Negra | • Míldio • Oídio (93,33%)
Sannakki et al. [26]	K-means Clustering	Textura: GLCM	Backpropagation Neural Network (BPNN)	• Míldio | • Oídio (100%)

Krithika e Selvarani criaram um identificador de doenças na vinha em que usaram *k-nearest neighbors* para a classificação, *tangential direction* para a parte da segmentação e para *feature extraction* optaram por usar *Gray-Level Co-Occurrence Method* (GLCM) e histogramas HSV e L*a*b* [19]. Hase et al. criaram um método de diagnóstico para descobrir se as doenças tinham como base um fungo ou uma bactéria. Para isto usaram *Gaussian Mixture Model* para ajudar na segmentação e *Fast Library for Aproximate Nearest Neighbors* (FLANN) para classificar as imagens através dos histogramas HSV extraídos de cada imagem. O resultado deste método foi de 93,2% para bactérias e 94,7% para fungos [20]. Elemmi et al. classificaram imagens de folhas de videira com diferentes distâncias e luminosidades, obtidas através de uma câmera digital de alta resolução, como sendo saudáveis ou tendo míldio, oídio, antracnose ou com base bacterial.

Tabela 3.2: - Informação sobre os Datasets

Referência	Nº de Imagens	Detalhes
Hase et al. [20]	100 (Bactéria) + 100 (Fungo)	Câmera
Elemmi et al. [21]	75 (Antracnose) + 72 (Bactéria) + 68 (Míldio) + 81 (Oídio)	Câmera Digital de alta resolução. Diferentes distancias e luminosidades.
Agrawal et al. [3]	40 (Podridão Negra) + 40 (Esca) + 40 (Leaf Blight) + 40 (Saudáveis)	Imagens obtidas em www.plantvillage.org. Imagens com alta resolução
Waghmare e Kokare [22]	160 (Saudável) + 290 (Infectadas)	Câmera de Telemóvel Formato: .jpg (2,5,8,13 MP)
Sandika et al. [23]	900 (Infectadas)	Imagens obtidas por agricultores (Maharashtra - Índia) Câmera de Telemóvel (<1 - 13 MP)
Padol e Yadav [18] (A) Padol e Sawant [24] (B)	75 (Míldio) + 62 (Oídio)	Câmera de Telemóvel (Pune e Nasik - Índia) + Imagens obtidas na web Formato: .jpg
Kharde e Hulkarni [25]	72 (Míldio) + 72 (Oídio) + 72 (Podridão Negra) + 72 (Saudável)	Iball USB Camera Formato: .jpeg (20 MP)
Sannakki et al. [26]	17 (Míldio) + 16 (Oídio)	Câmera digital (Nikon Coolpix P510) Formato: .jpg (16.1 MP)

Com este intuito usaram para a classificação uma *Support Vector Machine* (SVM) com os dados de cor, textura e forma extraídos das imagens. O resultado destas classificações foi de 96,6% [21]. Agrawal et al. classificaram imagens, com alta resolução, como tendo podridão Negra, esca ou *leaf blight* com uma percentagem de acerto de 90%. Para isto usaram *k-means clustering*, histogramas HSV e L*a*b* para a cor e GLCM para a textura, e sobre estes dados uma *Support Vector Machine*. As imagens foram obtidas em www.plantvillage.org [3]. Waghmare e Kokare usaram *Multi-Level Thresholding* e uma *Support Vector Machine* sobre histogramas HSV e texturas obtidas por *Opposite Colour Linear Binary Pattern* para diagnosticar as doenças míldio e oídio em imagens capturadas com câmera de telemóvel em formato .jpg. Foi obtida uma percentagem de acerto de 96,6% [22]. Sandika et al. obtiveram 86% de acerto no diagnóstico de míldio, oídio e antracnose em imagens de folhas capturadas por agricultores na Índia obtidas através de câmeras de telemóvel. Para isto usaram *Region of Interest* (ROI) e para a classificação *Random Forest*. Apenas foram extraídas características de textura (GLCM e Local Binary Pattern) [23]. Padol com Yadav e Padol com Sawant [24] criaram dois métodos de diagnosticar míldio e oídio em folhas de videira. Em ambos projectos foi usado *k-means clustering* para a segmentação e extraídas as mesmas características de cor (HSV) e de textura. No primeiro projecto para a classificação das doenças foi usado *Support Vector Machine* (SVM) e foi obtida uma eficácia de 93,3% e 83,3% para míldio e oídio respectivamente [18]. No segundo projecto foi adicionada uma *Artificial Neural Network* (ANN) à anterior SVM e esta aumentou a eficácia para 100% em ambas doenças [24]. Kharde e Hulkarni usaram o algoritmo *Watershed*, características de textura através de GLCM e uma *Artificial Neural Network* para diagnosticar podridão negra, míldio e oídio com um resultado de 93,33% [25]. Sannakki et al. usaram mais uma vez *k-means clustering* para a segmentação, extraíram mais uma vez características de textura através de GLCM e uma *Backpropagation Neural Network* (BPNN) para a classificação das doenças míldio e oídio. O resultado obtido com o *dataset* usado foi de 100% [26].

3.3 Discussão

Todos os artigos mencionados contém propostas interessantes na área de diagnóstico de doenças através de imagens das folhas. Todos estes projectos tem como base o esquema da Figura 3.1. Inicialmente passam pelo processo de **pré-processamento**. Este processo é mais ou menos necessário consoante as condições de captura das imagens a analisar. Imagens capturadas com reflexos e sombras irão necessitar mais pré-processamento pois estas condições podem causar ruído na altura de caracterizar a imagem. O seguinte passo é a **segmentação**, que num projecto deste tipo consiste em separar a folha do fundo em que esta se insere. Esta tarefa pode ser facilitada ao efectuar uma captura da imagem num ambiente controlado onde o fundo é facilmente distinguido da folha. Por outro lado um fundo com características semelhantes às presentes na folha irá dificultar esta tarefa. De seguida é efectuada a **extração de *features***, onde o objectivo é extrair da imagem já processada as melhores características/features que a definem. Como podemos ver nos projectos acima, existem vários tipos de *features* que podemos retirar de uma imagem, nomeadamente de cor, textura e forma, sendo que cada uma destas características podem ser definidas de diferentes formas. Podemos retirar características de cor através dos espaços de cor RGB, HSV e L*a*b. LBP, GLCM e Filtros Gabor são formas de definir a textura de uma imagem. O último passo é treinar o *dataset* já com todas as características através de um algoritmo de classificação, como por exemplo SVM, ANN, Random Forest ou KNN, entre outros. Este algoritmo irá associar a imagem que queremos classificar às já classificadas comparando as suas características.

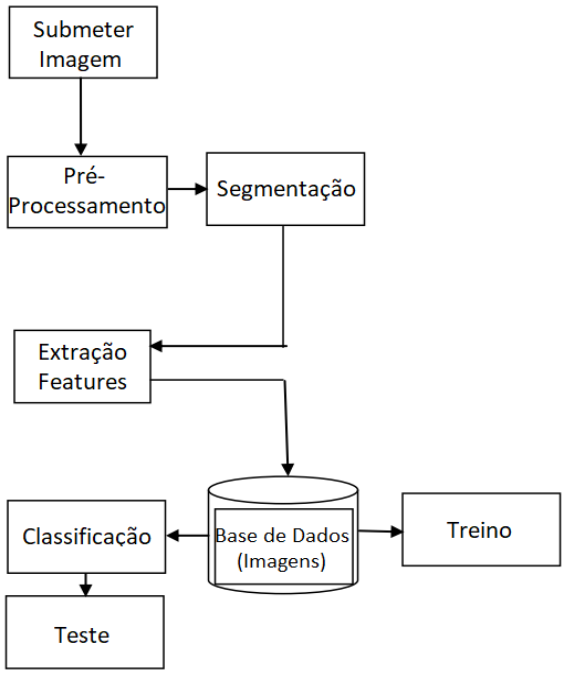

Figura 3.1: Esquema inicial do projecto [1]

[1] Adaptado de: [3]

Capítulo 4

Desenho e Desenvolvimento

Este capítulo está, como tinhamos dito anteriormente, dividido em três partes. Na primeira parte, são apresentados os *datasets* e ferramentas usadas, assim como o porquê desta escolha. Na segunda parte, explicamos como foi implementada cada uma das partes do sistema, incluindo as modificações efectuadas nas imagens (com figuras a demonstrar como actua em cada um dos três tipos de folha) e ainda a ligação entre cada uma das etapas. São também debatidos os problemas encontrados e a sua resolução. Finalmente, na última parte, mais uma vez discutimos o que foi aprendido neste capítulo através da implementação do sistema.

4.1 Desenho

4.1.1 Datasets e Doenças

Os *datasets* usados foram obtidos através de uma plataforma *online* e foram agregadas préviamente pela *PlantVillage* (https://plantvillage.psu.edu), uma unidade de pesquisa e desenvolvimento da Universidade da Pensilvânia que capacita pequenos agricultores e procura tirá-los da pobreza ao usar tecnologia barata e acessível e dando-lhes acesso ao conhecimento obtido para ajudá-los a cultivar mais alimentos. As imagens disponibilizadas são de folhas saudáveis, folhas infectadas com esca, ou infectadas com podridão negra. O *dataset* possui um total de 2985 imagens, divididas em 1179 com folhas infectadas com podridão negra, 1383 com folhas infectadas com esca, e finalmente 423 imagens de folhas saudáveis. Apesar deste desequilibrio das imagens de folhas saudáveis em comparação às outras, demonstramos nesta dissertação que isto não é um problema. As imagens foram captadas em ambiente controlado com um fundo em pedra cinza tal como as imagens da Figura 4.1. Apesar de as imagens terem sido captadas em ambiente controlado, possuem algum ruído que teve de ser retirado na étapa de pré-processamento. Este ruído advém das condições de luminosidade na altura da captura das imagens. Podemos observar em algumas imagens, reflexo de luz que camuflam algumas características que podem ser importantes na hora de diagnosticar uma certa doença. Por outro lado, desta luz advém também sombras que além de também camuflar algumas características também confundem na hora de fazer a segmentação,

pois distorcem as fronteiras dos píxeis de primeiro plano (folha), com os píxeis de fundo.

Figura 4.1: Imagens do dataset *PlantVillage* - folha saudável, folha com esca, folha com podridão negra (da esquerda para a direita)

4.1.2 Escolha das Ferramentas Utilizadas

A escolha das ferramentas utilizadas, na maioria dos casos, teve como motivos principais, a preferência do autor, assim como conselho por parte do orientador. Como IDE para o desenvolvimento do nosso sistema foi escolhido o *Microsoft Visual Studio* , devendo-se esta escolha unicamente ao facto de o *Visual Studio* ser o IDE mais utilizado na atualidade, como podemos verificar na Figura 4.2.

Worldwide, May 2019 compared to a year ago:

Rank	Change	IDE	Share	Trend
1		Visual Studio	22.81 %	-3.0 %
2		Eclipse	20.05 %	-4.1 %
3		Android Studio	19.4 %	+8.2 %
4		NetBeans	6.56 %	-0.1 %
5		IntelliJ	4.75 %	+0.3 %
6	↑↑↑↑	Visual Studio Code	4.5 %	+1.2 %
7	↑↑	pyCharm	4.43 %	+0.9 %
8	↓↓	Sublime Text	3.86 %	-0.5 %
9	↓↓	Atom	3.62 %	-0.7 %
10	↓↓	Xcode	3.35 %	-0.4 %

Figura 4.2: Ranking IDEs (Maio 2019)
[1]

[1] Adaptado de: https://pypl.github.io/IDE.html

Para aplicar técnicas de tratamento de imagem, juntamente com o *Visual Studio*, foi escolhida a biblioteca OpenCV. As duas linguagens de programação mais usadas juntamente com o OpenCV são C/C++ e Python das quais optamos por escolher a primeira por gosto pessoal.

Para treinar o algoritmo e posteriormente testá-lo foi escolhido o IDE RStudio que usa a linguagem R e é utilizado maioritariamente para realizar estatísticas e prever acontecimentos.

Para a programação da aplicação em Android foi escolhido o IDE Android Studio SDK. Esta escolha foi uma junção entre o gosto pessoal do autor e o facto de este IDE ser um dos melhores no que toca a programação em Android. A linguagem de programação usada no Android Studio foi Java.

4.2 Desenvolvimento

A interligação entre as ferramentas mencionadas acima é a existente na Figura 4.3. Para testar o funcionamento completo do sistema, optamos por criar uma aplicação *Android* e um servidor associados ao sistema. O processo começa então com a submissão de uma imagem da folha que queremos diagnosticar por parte do dispositivo *Android* na aplicação. Ao submeter a imagem, esta é enviada para o servidor e armazenada numa base de dados. Simultaneamente é enviado o URL onde foi armazenada a imagem, juntamente com o nome da própria imagem como argumentos para o executável do programa C++. Através destes argumentos o programa que processa a imagem consegue chegar à imagem que queremos diagnosticar. Apartir do momento que tem a imagem, já consegue começar com o primeiro passo que é o pré-processamento.

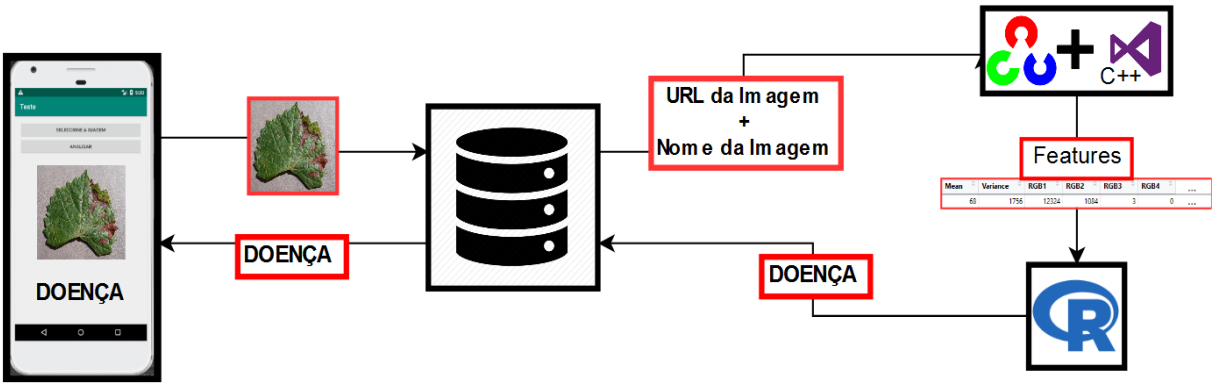

Figura 4.3: Arquitectura do Sistema

4.2.1 Pré-Processamento

Iniciamos então o tratamento das imagens com o pré-processamento. Como vimos anteriormente, o principal objetivo do pré-processamento é modificar uma imagem para que o resultado final seja mais adequado do que a imagem original para uma aplicação específica. Lida principalmente com a melhoria dos dados da imagem, ao suprimir ruídos indesejados e ao aprimorar algumas características importantes da imagem em questão. Após a análise do *dataset*, reparamos que existe ruído que advém das condições de luminosidade na captura da imagem. Estas condições trouxeram à imagem, reflexos e sombras. Como sabemos, o nosso objectivo neste pré-processamento é remover estes reflexos e sombras para deixar mais salientes as mazelas deixadas pela doença em questão.
Então para o pré-processamento, começámos por fazer um *Contrast Stretching* sobre a imagem original. Como vimos no Capítulo 2, *Contrast Stretching* consiste em esticar a faixa de valores de intensidade que a imagem contém para aproveitar ao máximo os valores possíveis. É frequentemente usado para aprimorar imagens de baixo contraste. No nosso programa, esta técnica foi implementada na função ***ContrastStretch*** e tem os resultados da figura abaixo.

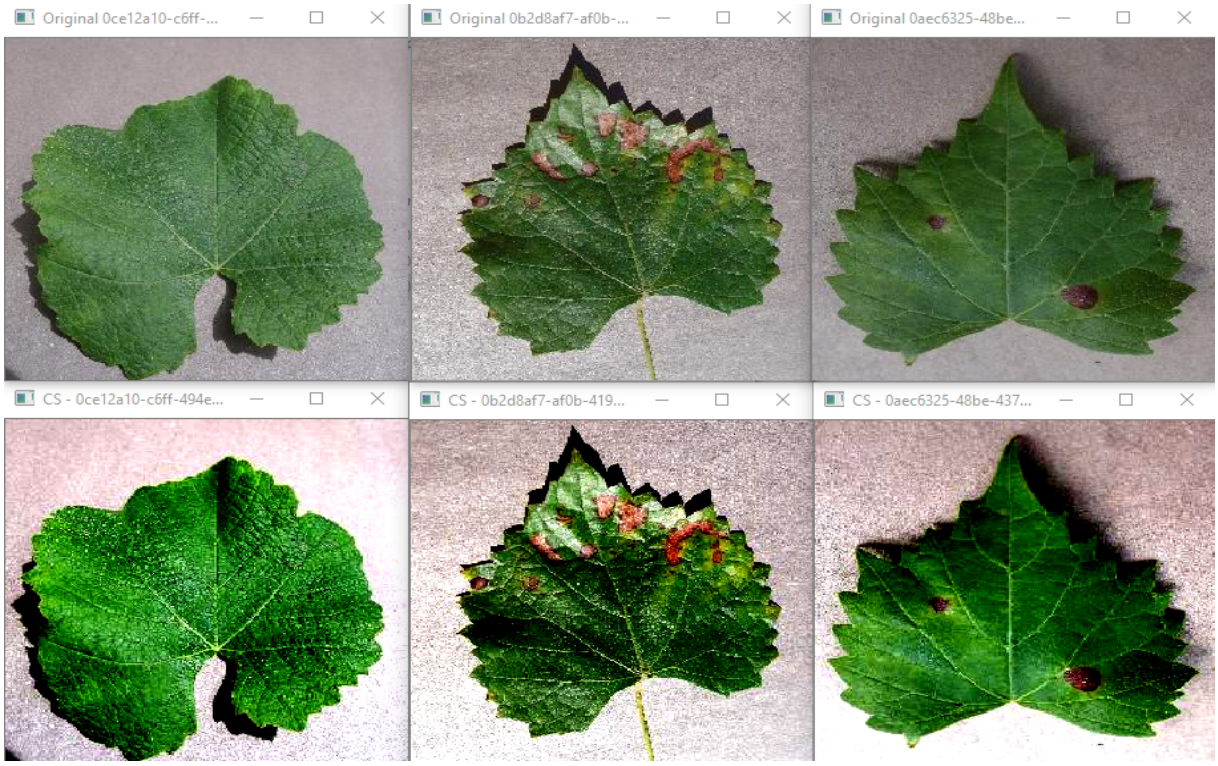

Figura 4.4: Contrast Stretching

Como podemos ver nas imagens exemplo, esta técnica aumenta o contraste, deixando mais saliente a diferença de cor entre os píxeis de primeiro plano (folha) e os de fundo.

De seguida passamos à fase de resolver ou pelo menos diminuir o problema que são os reflexos de luz nas folhas. Efectuamos uma série de processos na imagem, com vista a remover reflexos de luz. Inicialmente aumentámos a saturação da imagem para obter a versão mais pura da cor em questão. Esta técnica foi implementada na função ***saturationEnhancement*** e consiste em converter a nossa imagem no espaço de cores RGB para o espaço de cores HSV e colocar a componente de saturação S com o valor 200, para posteriormente voltar a converter a imagem para RGB. Na altura da implementação, este valor pareceu o mais indicado na hora de deixar mais salientes os píxeis cuja componente RGB dominante é o verde. Desse resultado colocámos a zero (preto) todos os píxeis em que o valor de verde é menor que o valor de vermelho ou azul, ou então os píxeis onde os valores dos três componentes são superiores a 60. Mais uma vez, foi através da observação que concluímos que estes eram os melhores valores a aplicar, com vista a manter apenas as partes verdes da folha. Esta operação foi implementada na função ***mantainGreen***.

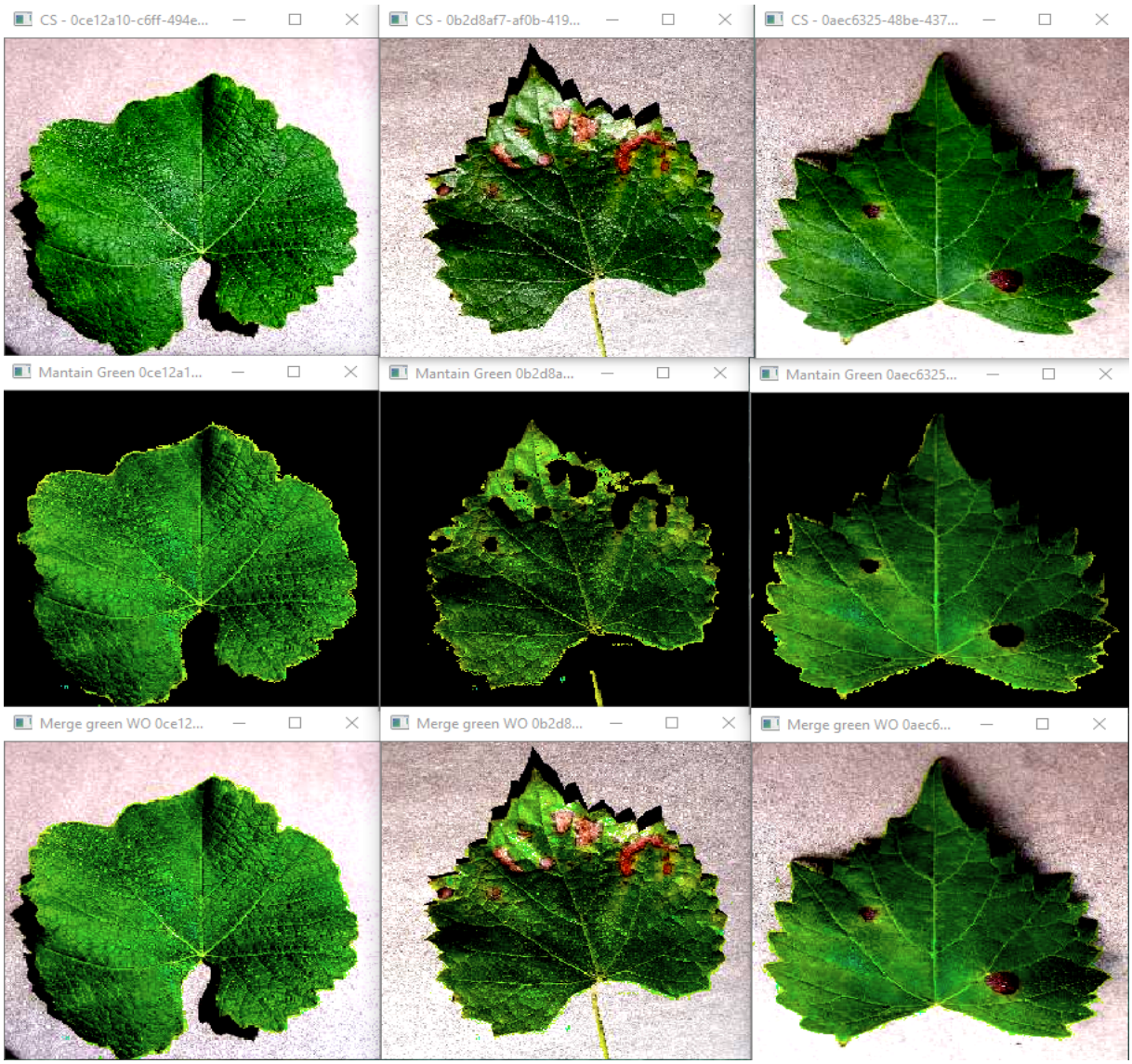

Figura 4.5: Eliminação de Reflexos

Por fim fazemos uma junção da imagem, já com *contrast stretching* efectuado (Figura 4.5 - 1ª linha), com a imagem resultante do aumento da saturação (Figura 4.5 - 2ª linha). Esta operação dá o resultado vísivel na 3ª linha da Figura 4.5. Esta operação de junção está implementada na função ***mergeGreenWithOriginal***. Nesta função percorremos os píxeis das imagens da 2ª linha e no caso do píxel ser diferente de zero, é mantido o píxel dessa imagem, em caso contrário o píxel toma o valor do píxel da imagem da 1ª linha na mesma posição. Como podemos observar houve uma ligeira redução dos reflexos, sendo mais visíveis as diferenças entre a 2ª coluna das linhas 1 e 3 da Figura 4.5.

Para finalizar este pré-processamento, temos como objectivo identificar as sombras. Com isto em mente, convertemos novamente a imagem, resultante do processo anterior, para o espaço de cores HSV, visto que esta fica com as sombras mais salientes nesse espaço de cores mencionado. De seguida, juntámos a imagem RGB que temos até agora com a imagem HSV, sendo que sempre que os píxeis da imagem HSV sejam pretos, os mesmos píxeis na imagem RGB são colocados também a preto, obtendo o resultado da Figura 4.6 (nesta imagem colocámos as sombras a vermelho para serem mais identificáveis ao olho humano). Este processo foi implementado na função ***mergeTwo***.

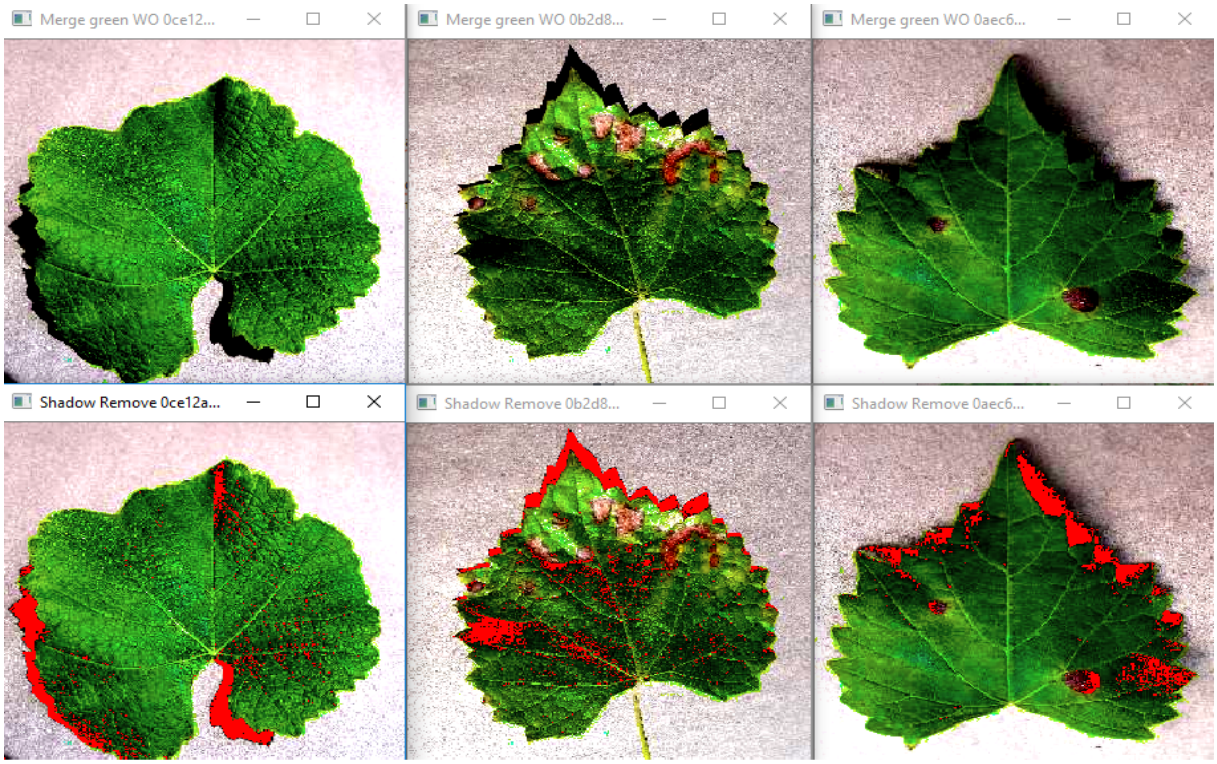

Figura 4.6: Identificação de Sombras - Aplicação da função *mergeTwo*

Tendo sido, removido algum ruído de reflexos de luz e identificada a maioria das sombras, podemos passar para a seguinte fase, que é a segmentação. Relembrando que o pré-processamento não é 100% eficaz e que continua a passar algum ruído especialmente nos casos em que houve piores condições no processo de captura da imagem.

4.2.2 Segmentação

O próximo passo correspondente à segmentação, que relembrando o Capítulo 2, consiste em sub-dividir uma imagem em diferentes áreas, regiões ou objectos. O nível ao qual esta sub-divisão deve ser levada depende do problema em questão. Neste caso optamos por efectuar uma segmentação de dois níveis, onde apenas separámos os píxeis que correspondem à folha dos píxeis que correspondem ao fundo, supondo assim que as sombras fazem parte do fundo. Relembramos também que a segmentação de imagens não triviais é uma das tarefas mais difíceis no processamento de imagens. A precisão da segmentação determina o eventual sucesso ou fracasso dos próximos passos do sistema. Daí ter uma boa percentagem de acerto na hora de efectuar a segmentação é mandatório para o sucesso dos próximos passos e para a eficácia do diagnóstico em geral.
Então, no processo de segmentação optámos por usar a técnica de *thresholding* para identificar que píxeis da imagem correspondem ao fundo no qual a folha se encontra inserida, e quais os que correspondem à folha em si, tal como mencionado acima. Relembrando o que foi dito no Capítulo 2, as técnicas de *thresholding* podem ser divididas em *bi-level* e *multi-level*, dependendo do número de segmentos da imagem. Como neste caso queremos distinguir apenas fundo de folha, usaremos o *bi-level thresholding*, onde a imagem é segmentada em duas regiões diferentes. Após exaustiva observação, chegámos à conclusão que os píxeis com valores de R, G e B maiores que $T = 50$ podem ser classificados como píxeis pertencentes ao fundo, e os restantes píxeis com valores menores que o próprio T como fazendo parte da folha.

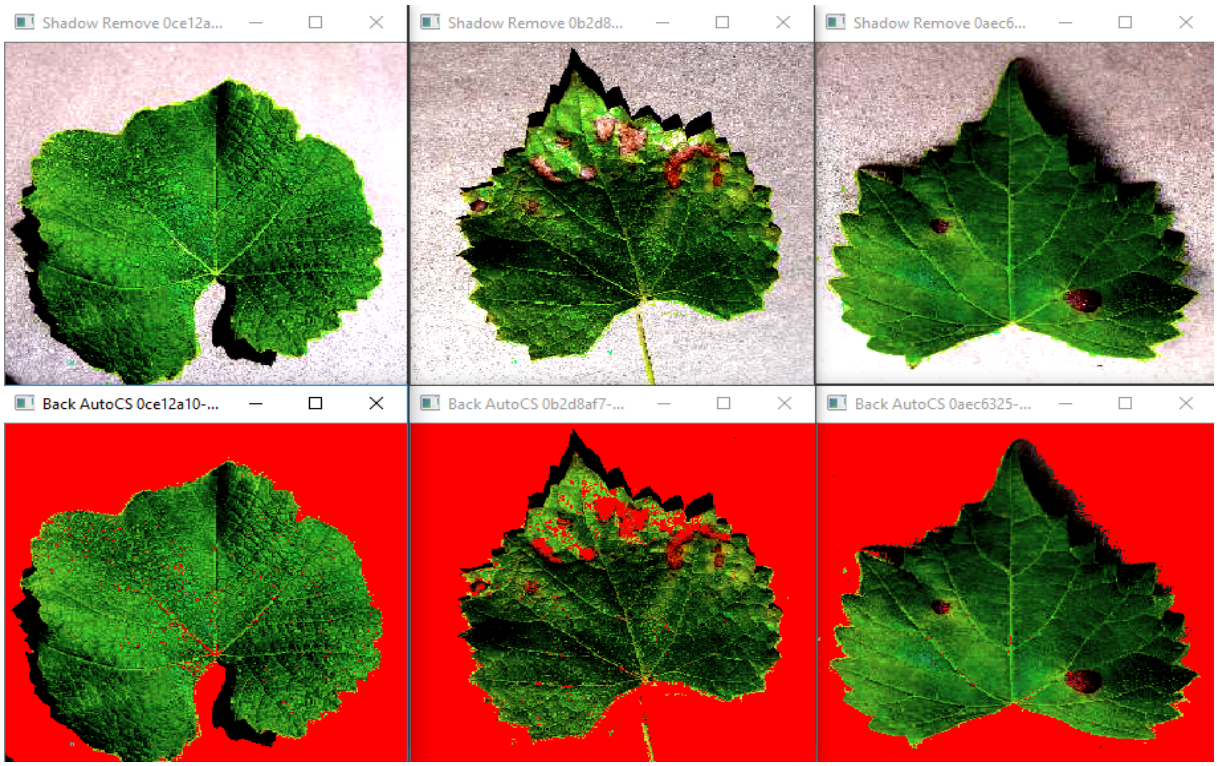

Figura 4.7: Identificação dos píxeis de fundo - Aplicação da função *binaryImage/threshold* (Vermelho)

Os píxeis classificados como fazendo parte do fundo são os que aparecem a vermelho na Figura 4.7. Esta identificação dos píxeis de fundo foi implementada na função ***removeBack***, mas em vez de vermelho, os píxeis de fundo foram colocados a preto tal como os píxeis de sombra. E o resultado é o apresentado na Figura 4.8.

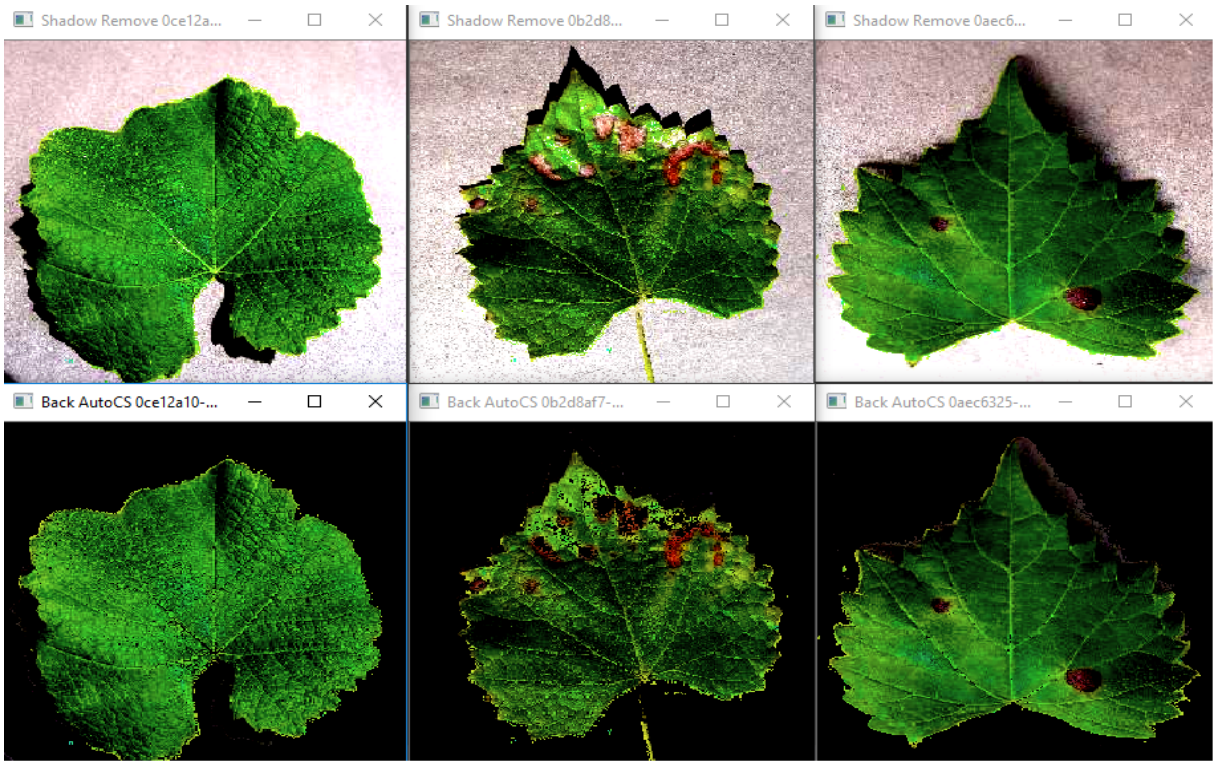

Figura 4.8: Identificação dos píxeis de fundo - Aplicação da função *removeBack/threshold* (Preto)

Como podemos observar ainda restou algum ruído neste passo de segmentação. Este ruído é visível ao nível da fronteira entre as regiões de sombra da folha e as regiões de fundo. Este ruído deve-se à fronteira não ser constante e apresentar alguns píxeis de preto distantes uns dos outros na zona do fundo de pedra, como é evidente na 3ª coluna da Figura 4.8. Outra fonte de ruído são as diferenças de textura no interior da folha.
Com o objectivo de remover ou diminuir este ruído, executámos uma série de passos. O primeiro foi converter a imagem que temos até ao momento numa imagem binária, sendo que todos os píxeis que não são pretos, são colocados a branco, obtendo o resultado da Figura 4.9. O nome binário advém da numeração binária onde apenas se usa 0 e 1. Neste caso consideramos os píxeis pertencentes à folha como sendo píxeis com valor 1 e todos os outros como sendo de valor 0. A esta função demos o nome de ***binaryImage***.
Como podemos observar na Figura 4.9, nas imagens binárias é mais saliente o ruído que provém das sombras na textura do interior da folha. Com o objectivo de remover a maior parte deste ruído passamos para o próximo passo desta segmentação.

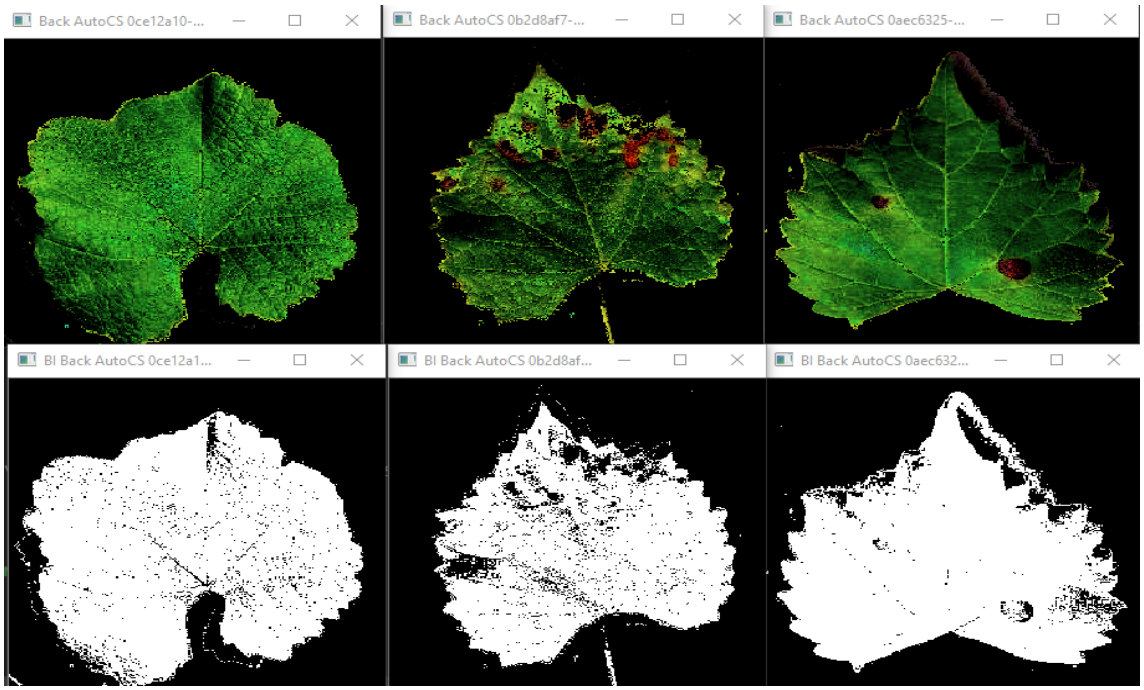

Figura 4.9: Conversão da imagem para binário - Aplicação da função *binaryImage*

O seguinte passo, que consideramos oportuno aplicar, foi um filtro morfológico, de modo remover o ruído visível no interior da folha, representado pelos pequenos pontos negros, mais conhecidos por *pepper noise*. Relembrando o que foi dito no Capítulo 2, os filtros morfológicos, podem ser aplicados para aumentar e diminuir regiões da imagem, assim como para remover ou preencher píxeis nas fronteiras entre as regiões da imagem.

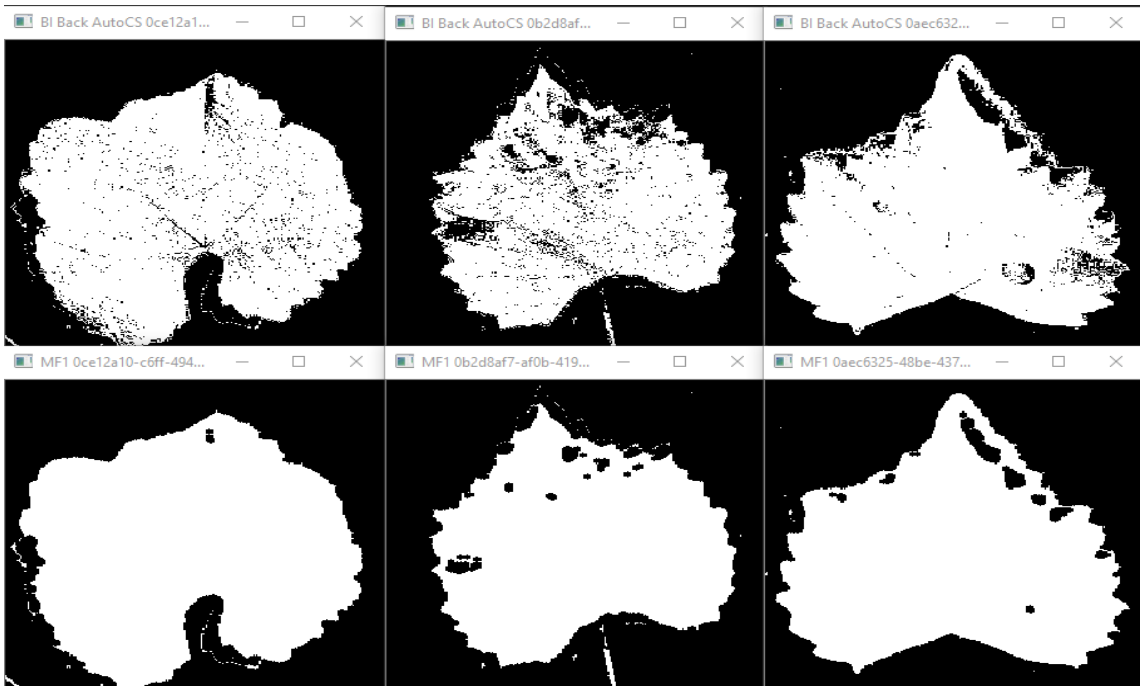

Figura 4.10: *Threshold* do Fundo

Decidimos aplicar um filtro morfológico de *close*. O motivo para esta escolha, é por este ser o ideal para preencher os pequenos orifícios de fundo conhecidos como pepper noise, tal como dissemos no Capítulo 2. Este tipo de operação tende ainda a ampliar os píxeis dos limites das regiões de primeiro plano. À função que aplica o filtro morfológico à imagem binária demos o nome de ***morphFilter*** e depois de tentar vários elementos estruturais decidimos que o que mais se adequa, pois apresenta melhores resultados é a *ELLIPSE*. O quadrado, que é o elemento estrutural padrão fazia com que as fronteiras das folhas ficassem muito forçadas e pouco reais. O resultado da aplicação desta função é apresentado na Figura 4.10. Como podemos observar nas imagens a maior parte do *pepper noise* desapareceu. Podemos passar então para o último passo da nossa segmentação que é a obtenção da nossa imagem, novamente em RGB.

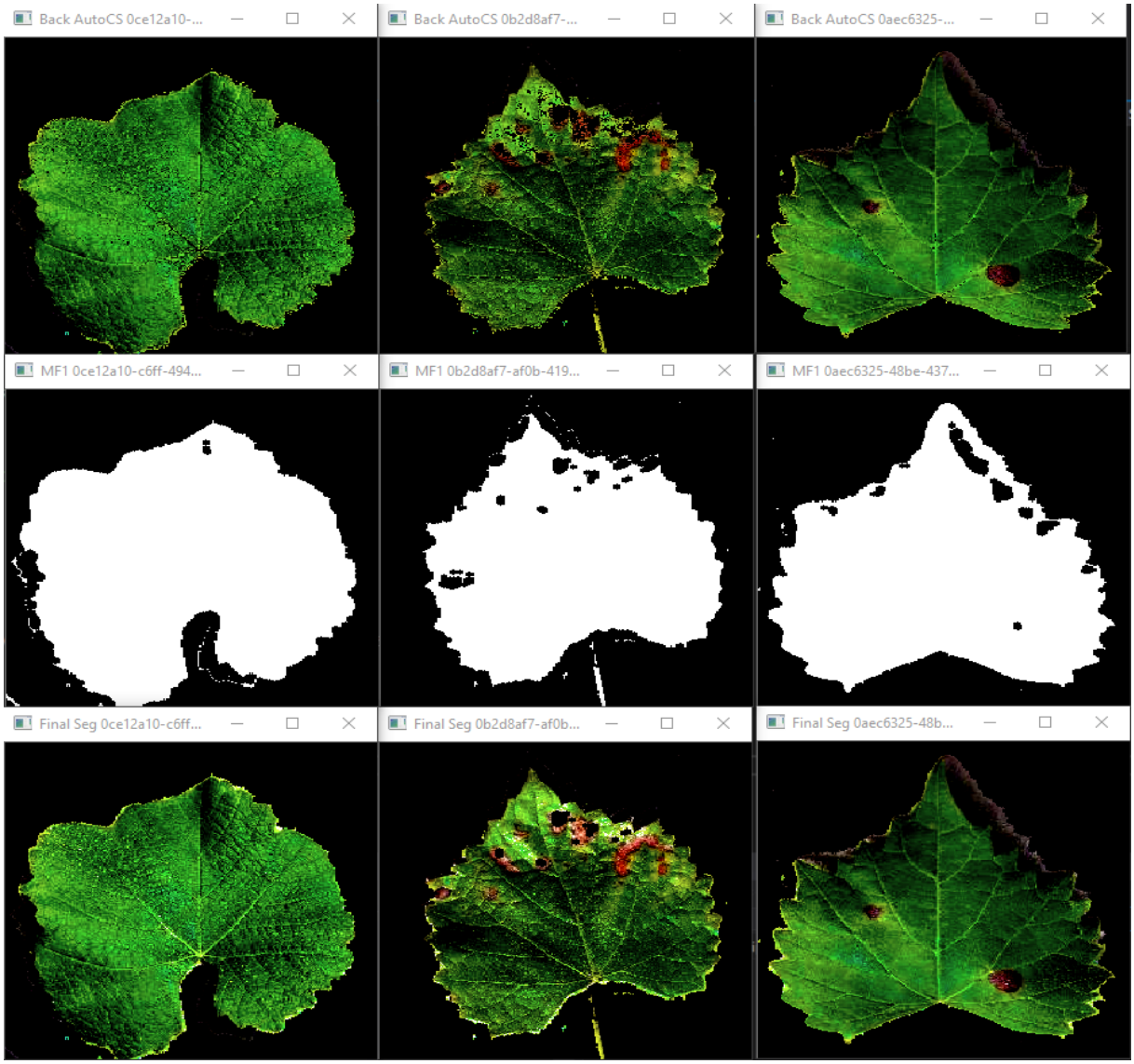

Figura 4.11: Obtenção da imagem final segmentada

Neste passo, voltamos a usar a função ***mergeTwo***. Nesta função, juntámos a imagem processada até ao momento (1ª linha da Figura 4.11), com esta imagem binária à qual aplicamos o filtro morfológico (2ª linha da Figura 4.11), obtendo a nossa segmentação final. Como podemos observar nas folhas da 3ª linha da Figura 4.11, houve uma redução bastante significativa no ruído existente nas imagens segmentadas, nomeadamente no *pepper noise*. No entanto, não deixamos de assinalar que a segmentação não coincide na totalidade à segmentação real, com as fronteiras exactas. No Capítulo 5 fazemos uma comparação onde calculámos a percentagem de semelhança entre esta segmentação e uma outra realizada manualmente em algumas imagens do *dataset*.

4.2.3 *Feature Extraction*

O próximo passo do nosso sistema é o *feature extraction*. Este passo, como foi dito no Capítulo 2, consiste em descrever as informações relevantes contidas numa imagem. Estas informações podem ser apresentadas por descritores de vários tipos. Existem descritores de cor, textura e forma. O nosso objectivo é aproveitar os melhores descritores para descrever da melhor forma possível as imagens, dependendo dos sintomas que cada uma possua. Ao sabermos que uns determinados sintomas tem determinadas características, tanto de cor, como de textura ou forma, quando estivermos a diagnosticar uma nova imagem, que possui as mesmas características, vamos chegar à conclusão que sofrem da mesma doença. No nosso sistema começamos por implementar descritores de cor para as imagens.

4.2.4 Cor

Começamos então por implementar um descritor para extraír características sobre o espaço de cores RGB das imagens. Nesta altura decidimos que a melhor forma para descrever um espaço de cores é através de um histograma.

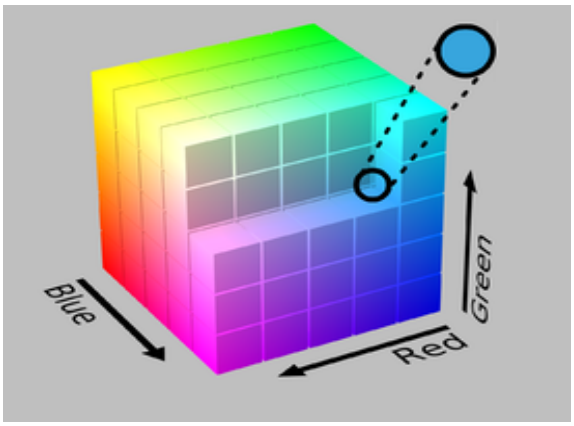

Figura 4.12: Cubo RGB
[2]

[2]Adaptado de: https://upload.wikimedia.org/wikipedia/commons/thumb/8/83/RGB_Cube_Show_lowgamma_cutout_b.png/1280px-RGB_Cube_Show_lowgamma_cutout_b.png

À função que cria o histograma que descreve o espaço de cores RGB, demos o nome de ***RGBHistogram***. Para perceber como é criado este histograma imaginemos o cubo da Figura 4.12, onde dividimos cada cor em 5 partes. Então, tendo 5 partes para Red, 5 para Green e 5 para Blue, possuímos no total 125 coeficientes para os valores de RGB. Em cada um destes coeficientes estará o número de píxeis da imagem com valores de R, G e B dentro de um determinado intervalo, não necessariamente igual. No primeiro dos coeficientes estará o número de píxeis da imagem com valores R = 0 , G = 0 e 1 ≤ B < 52, sendo estes píxeis próximos da cor preta. Optámos por excluir os píxeis com todos os componentes a zero pois correspondem aos píxeis pertencentes ao fundo ou à sombra, e não descrevem a cor presente na folha. Por outro lado, no último dos coeficientes estará o número de píxeis com valores 205 ≤ R < 256, 205 ≤ G < 256 e 205 ≤ B < 256, ou seja píxeis próximos da cor branca.

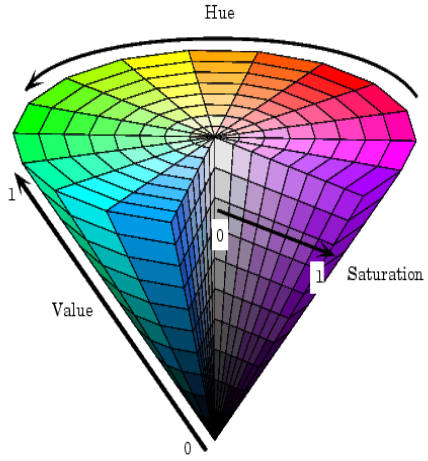

Figura 4.13: Cone HSV
[3]

De seguida passamos a criar os histogramas que descrevem as características do espaço de cores HSV das imagens. À função que cria o histograma demos o nome de ***HSVHistogram***. Imaginemos agora o cone HSV da Figura 4.13, optamos por dividir as componentes em 16 partes para o *Hue*, e 4 partes tanto para o *Saturation* como para o *Value*, obtendo um total de 256 coeficientes para cada histograma. Após exaustiva investigação , concluímos que esta divisão é a mais comum neste tipo de espaço de cores. Tendo em conta o processo realizado anteriormente para o espaço de cores RGB, mais uma vez temos, no primeiro coeficiente, o número de píxeis onde H = 0, S = 0 e 1 ≤ V < 64. Por outro lado, no último dos coeficientes estará o número de píxeis com valores 240 ≤ H < 256, 192 ≤ S < 256 e 192 ≤ B < 256. Neste processo de associar cada píxel ao seu respectivo coeficiente de valores HSV, optámos por eliminar os píxeis com *Value = 0* pois são píxeis de cor preta, ou seja, correspondem a fundo ou sombra, sendo assim não fazem parte da folha, então as suas características mais uma vez não nos interessam. Tendo já estes dois descritores de cor, histograma RGB e histograma HSV, podemos passar para a implementação dos descritores de textura.

[3]Disponível em: ps://pdfs.semanticscholar.org/cbae/0cf18596ea501962ac93d77d50fe2ff91e6d.pdf

4.2.4.1 Textura

Passámos então para a implementação do descritor de textura. Para extrair este tipo de características optámos por implementar um *Local Binary Patterns* (LBP). Revendo o que foi dito no Capítulo 2, esta técnica descreve cada píxel pelo nível de cinza relativamente aos seus píxeis vizinhos. Neste caso, optamos novamente por construir um histograma pois, continuamos que é a melhor forma de resumir as características de textura da imagem. Então começámos por converter a nossa imagem RGB, já segmentada, numa imagem em escala de cinza. Seguidamente, aplicamos o algoritmo para calcular o LBP para cada um dos píxeis da imagem. Este algoritmo foi implementado na função ***LBP***. Este algoritmo consiste em somar um certo valor, sempre que o tom de cinza de um píxel vizinho seja superior ao tom de cinza do píxel central.

1	2	4
8	Central	16
32	64	128

Figura 4.14: Esquema LBP

Os valores a somar são os apresentados na Figura 4.14, sendo que, se todos os píxeis vizinhos do píxel central corresponderem a píxeis com tons de cinza superiores a este, o algoritmo irá somar todos estes valores e resultar num máximo de 256, valor este que ocupará o lugar do píxel central na nova imagem LBP, como na Figura 4.15.

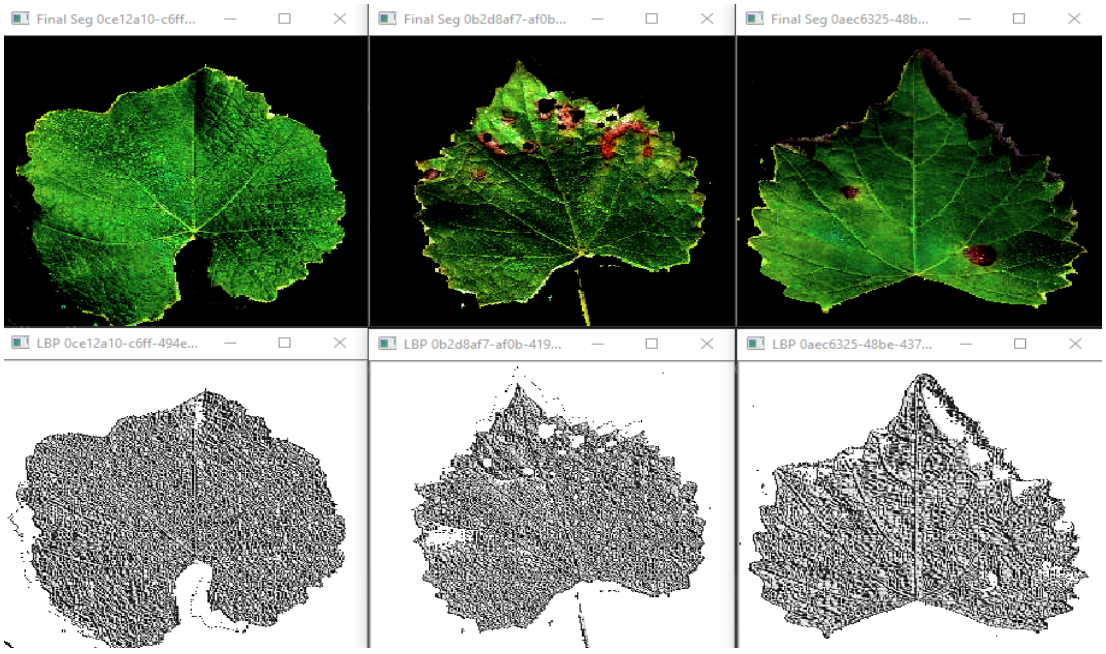

Figura 4.15: Imagem LBP - Aplicação da função *LBP*

No caso do histograma LBP, que implementamos na função ***get_256_Histogram***, que percorre todos os píxeis da imagem e verifica qual o valor do tom cinza, este valor varia entre 0 e 255, então irá ser colocado no respectivo coeficiente, ficando assim cada coeficiente com o número de píxeis com tom de cinza desse valor. Assim tendo finalizado o descritor de textura podemos passar para a implementação de um descritor de forma.

4.2.4.2 Forma

Para a obtenção das características de forma pensamos numa técnica para identificar a que distância do centro se encontra a fronteira entre a folha e o fundo. Para isto temos que ter em consideração vários angulos de busca. A esta função demos o nome de ***checkBorder***. O primeiro passo é converter, novamente, a imagem que temos até ao momento em binária, e de seguida traçar uma linha desde o centro da imagem até às várias direções que pretendemos, conforme apresentado na Figura 4.16. Obtemos então 8 coeficientes com o número de píxeis desde o centro da imagem até encontrar o primeiro píxel de cor preta, nas 8 direções.

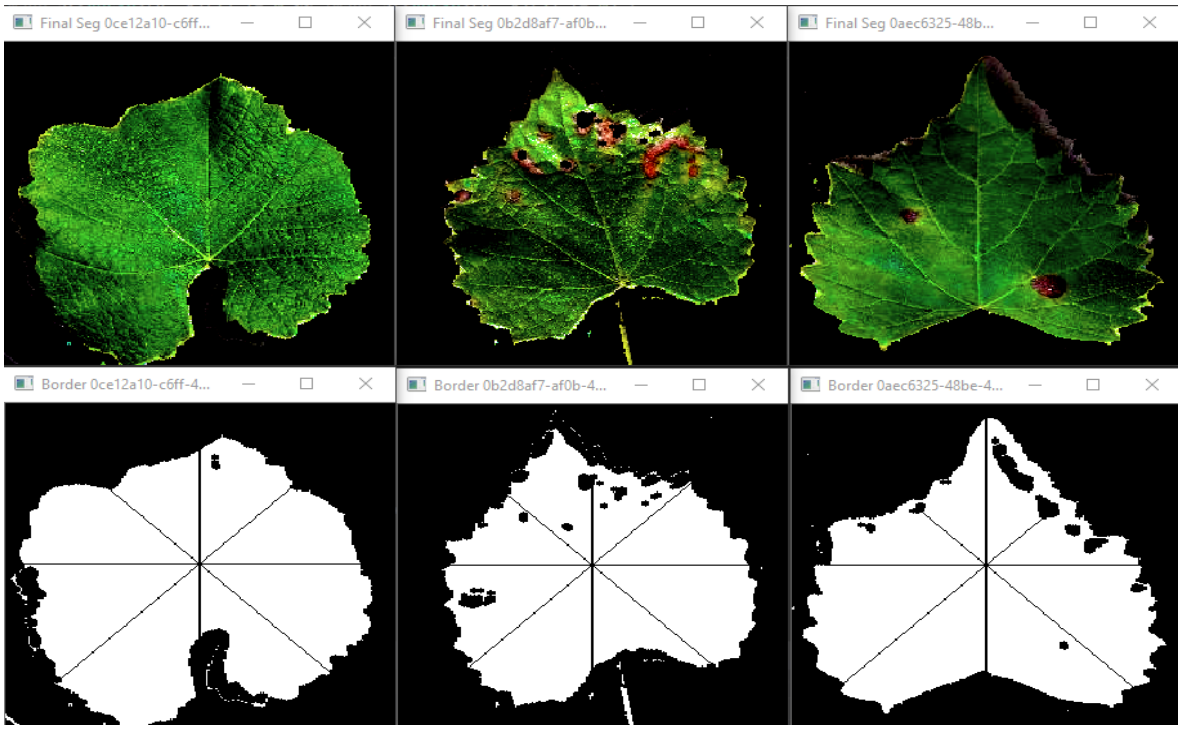

Figura 4.16: Exemplo de cálculo da distância à fronteira

Esta característica pode ser um bocado corrompida por ruído que tenha passado a fase de segmentação, pois se encontrar um píxel preto no interior da folha na direção pretendida irá assumir que este corresponde à fronteira. Por outro lado, a forma pode não ser um bom critério de distinção entre as várias doenças do nosso *dataset*. Tal como vimos no Capítulo 2, os sintomas das doenças do nosso *dataset* manifestam-se maioritariamente na cor. Daí a nossa previsão inicial, de que os descritores de cor devem ser os que melhor diferenciam as características das diferentes doenças.

4.2.5 Classificação

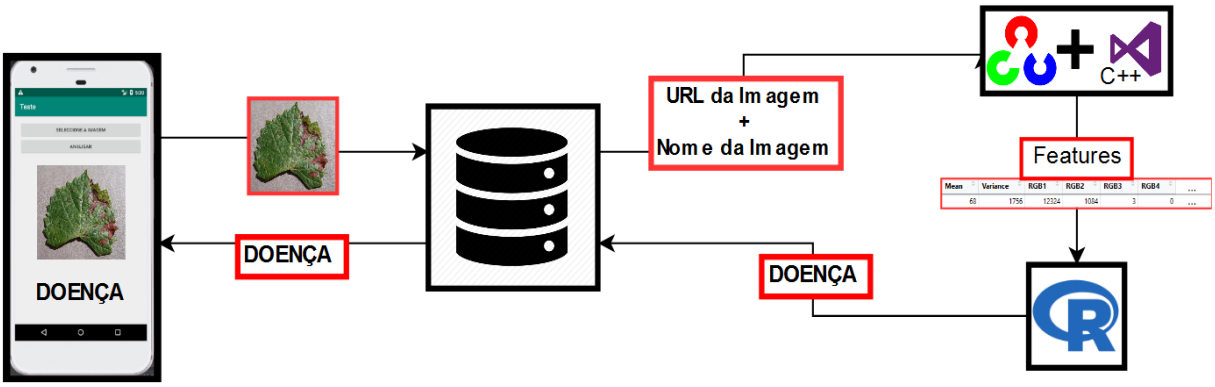

Figura 4.17: Arquitectura do Sistema

Como podemos ver na Figura 4.17, quando todas as *features* são extraídas, estas são acedidas por parte do classificador SVM implementado no *software RStudio*. Como é possível observar no Capítulo 2, uma *Support Vector Machine* é um classificador que usa um hiperplano entre duas classes que tenta dividir, uma das classes que contém o vector de treino rotulado como +1 da outra classe que contém o vector de treino rotulado como -1. Com o uso destes vectores de treino rotulados, o SVM encontra um hiperplano que maximiza a margem que separa as duas classes. Começámos então por treinar a nossa amostra com o algoritmo SVM, sendo que neste caso temos 3 classes. Quando este treino estiver terminado, significa que já existe um hiperplano que separa da melhor forma possível cada um dos possíveis diagnósticos, sendo estes "healthy"(folha saudável), "blackrot"(folha com podridão negra), ou "esca"(folha com esca). De seguida, são submetidas as *features* da folha que queremos diagnosticar, e através do algoritmo treinado anteriormente o sistema dá-nos o seu diagnóstico.
Tal como podemos observar, novamente na Figura 4.17, este diagnóstico é enviado para a base de dados e armazenado juntamente com os dados anteriores da imagem, tais como o URL de armazenamento, e o nome da imagem. O resultado do diagnóstico é ainda enviado como resposta para a aplicação no dispositivo *Android*, tal como podemos observar na Figura 4.18, de modo a que o utilizador execute as medidas de cura e prevenção adequadas à doença identificada.

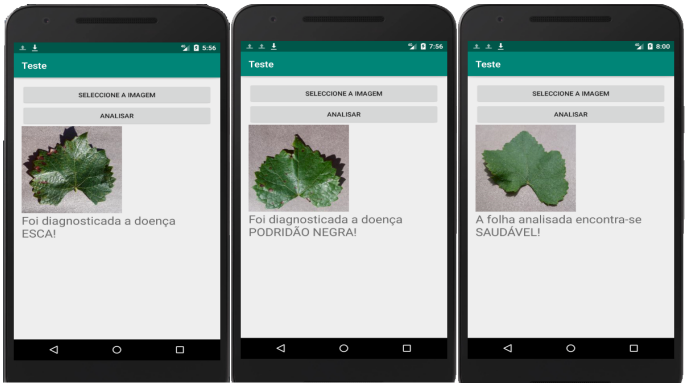

Figura 4.18: Comunicação de diagnóstico na aplicação *Android*

4.3 Discussão

Apesar do ruído presente nas imagens do *dataset*, conseguímos de uma forma ou outra dar a volta por cima e retirar características específicas a cada doença. Apesar de termos noção que algumas das técnicas não completaram a sua tarefa a 100%, no capítulo seguinte demonstrámos que atingimos valores bastante elevados, tendo em conta as condições com que as imagens do *dataset* foram capturadas, nomeadamente com reflexos e sombras. As sombras foram o aspecto que mais trabalho deu na hora de realizar a segmentação, pois algumas destas sombras estão presentes na folha, enquanto outras estão presentes no fundo. Isto, pode não induzir um diagnóstico errado, mas omite informação, que pode ser vital na hora de fazer o dito diagnóstico. Contudo, como vamos demonstrar, a segmentação conseguida, tem uma precisão também bastante elevada. Conseguímos implementar diferentes tipos de descritores, sendo assim possível comparar, como cada um deles actua sobre as imagens e a diferença de precisão dos diagnósticos efectuados por estes. Em comparação com os objectivos traçados inicialmente, conseguímos completar uma grande parte nesta etapa de desenvolvimento.

Capítulo 5

Testes e Resultados

Neste capítulo, apresentamos, como tinhamos dito anteriormente, uma comparação do processo de segmentação do nosso sistema, com uma outra segmentação realizada manualmente. É ainda feita uma análise à precisão na hora de diagnosticar correctamente as doenças presentes nas folhas por parte de cada tipo de descritor das imagens, bem como qual a combinação destes descritores que apresenta melhor resultado. Finalmente discutimos os resultados obtidos.

5.1 Segmentação

Com o intuito de validar a segmentação efectuada pelo nosso sistema, optámos por realizar uma segmentação manual através do *software Sefexa* em algumas das imagens do *dataset*. Depois comparamos estes dois tipos de segmentação através do coeficiente de *Dice*.

Coeficiente de Dice

Segundo Zou et al. [27], o coeficiente de similaridade de Dice é um índice de sobreposição espacial e uma métrica de validação de reprodutibilidade. O valor do coeficiente de Dice varia de 0 (0%), indicando que não há sobreposição espacial entre dois conjuntos de resultados binários, para 1 (100%), indicando a sobreposição completa. Neste caso, de comparação das diferentes segmentações, consideramos como sobreposição os píxeis que na mesma posição nas duas imagens estejam a branco. E dividimos este valor pelo número de píxeis que se encontram a branco numa imagem ou na outra, tal como podemos ver na seguinte expressão:

$$\text{Coeficiente de Dice} = \frac{\sum(\text{píxeis brancos em A} \cap \text{píxeis brancos em B})}{\sum(\text{píxeis brancos em A} \cup \text{píxeis brancos em B})}$$

À função que executa esta tarefa demos o nome de ***dice***. Como podemos ver na Figura 5.1, para as 3 imagens exemplo, correspondentes à folha saudável e a cada uma das doenças, a segmentação efectuada manualmente difere da segmentação efectuada automaticamente pelo sistema. Esta diferença deve-se maioritariamente ao ruído que passou nos processos, tanto de pré-processamento, como de segmentação. Acrescentado a esta diferença, está ainda o facto de a segmentação manual ter sido efectuada com recurso ao rato, em vez de uma outra ferramenta mais prática para o desenho das fronteiras. Daí a segmentação manual não coincidir 100% com a real. No entanto, o coeficiente de semelhança de Dice, é bastante alto, na ordem dos 90% para as 3 imagens exemplo. Foi calculada uma semelhança de 95,33%, 91,52% e 91,34% para cada uma das imagens da Figura 5.1, pela respectiva ordem.

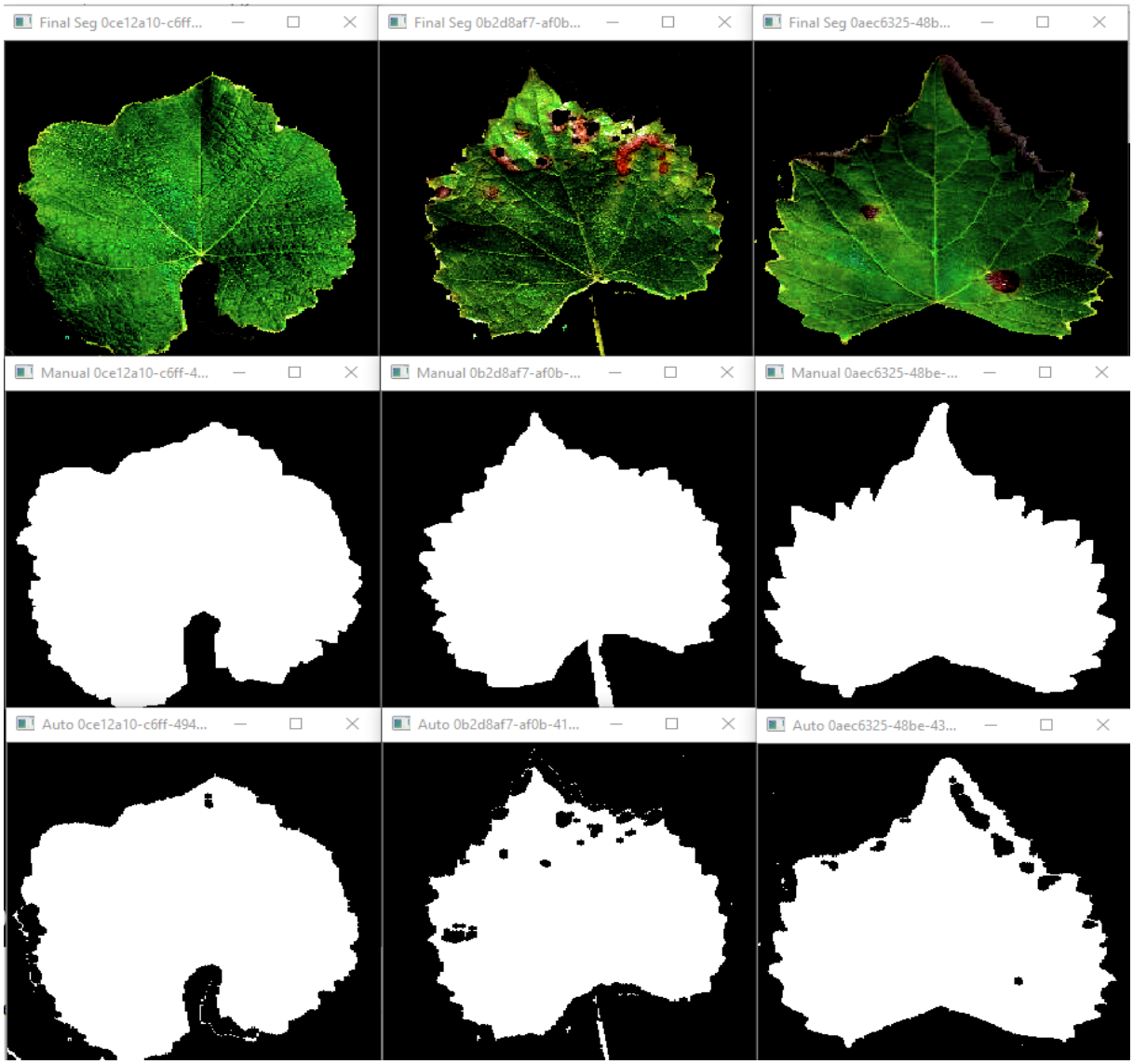

Figura 5.1: Segmentação Sefexa VS Segmentação Auto

Para ter uma melhor noção da qualidade da nossa segmentação, realizámos uma segmentação manual para mais 15 imagens, 5 por cada doença e calculámos o respectivo coeficiente de Dice. É possível observar o resultado no gráfico da Figura 5.2.

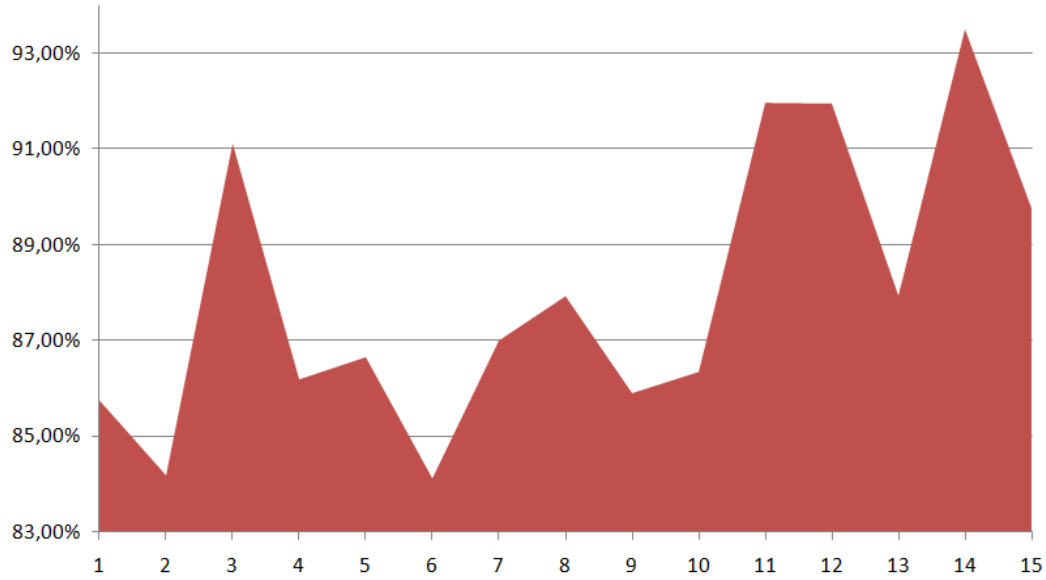

Figura 5.2: Coeficiente de semelhança de Dice

Como podemos ver no gráfico temos casos em que a segmentação não chega aos 85%, enquanto que noutras, passam dos 93%, ou 95% como numa das imagens exemplo. O aumento da percentagem é directamente proporcional à quantidade de ruído que passou pelas fases anteriores.

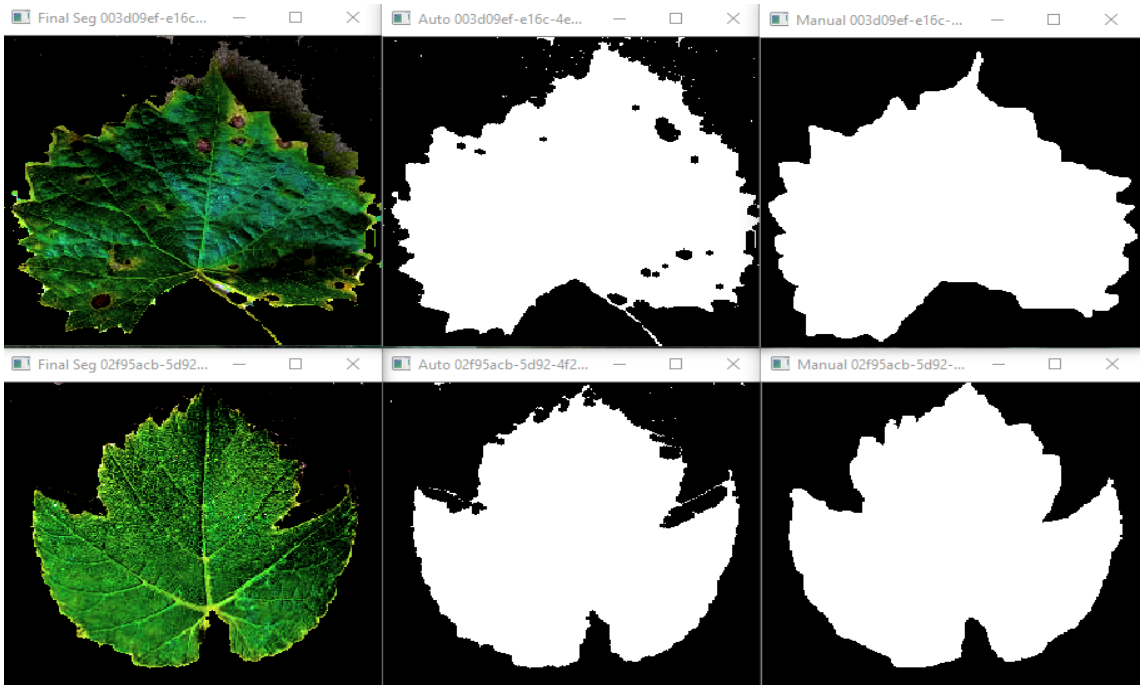

Figura 5.3: Em cima, o pior caso (imagem 2/15), e em baixo o melhor caso (14/15)

Com isto em mente, podemos concluir que a imagem 2 é aquela que possui mais ruído de entre as 15, enquanto que a 14 é a que menos ruído possui. Esta afirmação pode ser comprovada na Figura 5.3, onde podemos observar o pior, e o melhor caso de entre as 15 imagens às quais o gráfico da Figura 5.2 faz referência. Como podemos reparar, na imagem do pior caso, houve bastante ruído a passar as fases anteriores de processamento, enquanto que no melhor caso o ruído é quase nulo. Chegámos com isto à conclusão de que as condições de captura de imagem do melhor caso foram melhores do que as condições das outras.
No entanto, no geral consideramos os valores dos coeficientes de semelhança bastante aceitavéis, pois mesmo nos piores casos as percentagens andam à volta dos 80%, valor este, na nossa opinião, bastante alto.

5.2 Precisão de Diagnóstico

Passámos agora a analisar os vários descritores de imagem e a sua precisão na hora de fazer um diagnóstico. Para testar o nosso classificador com os vários tipos de *features*, optámos por fazer uma divisão de treino/teste de 90%/10%. A decisão recaiu sobre este 90/10 pois, apesar de 10% para a amostra de teste parecer pouco, no nosso *dataset* corresponde a aproximadamente 300 imagens, o que consideramos um número bastante razoável e capaz de abranger bastantes imagens de cada doença de forma a ser um teste proporcional. Para testar a precisão do algoritmo, em mais do que uma amostra, realizamos um *random subsampling*, também conhecido como *MonteCarlo cross validation*, que consiste em dividir o nosso dataset em várias amostras de treino/teste, todas elas com as percentagens acima mencionadas. Optámos por dividir então em 10 amostras, pois parece um número a partir do qual já conseguimos concretar entre que valores se repartem as precisões.

5.2.1 Espaço de cores RGB

Começamos então por analizar os nossos descritores de cor. Implementamos dois descritores de cor, porque na nossa opinião é a característica que mais diferencia as imagens na altura de fazer um diagnóstico. Este tipo de descritor funciona melhor na altura de distinguir as folhas saudáveis de todas as outras, pois estas apenas possuem diferentes tons de verde, enquanto que as outras possuem mazelas castanho-avermelhadas ou vermelho-amareladas. Iniciámos então por analisar a eficácia do histograma RGB como descritor das imagens, e dos sintomas das doenças em si.

		Previsão		
		blackrot	esca	healthy
Real	blackrot	105	5	0
	esca	15	136	0
	healthy	2	0	36

Figura 5.4: Matriz de confusão do descritor - Histograma RGB

A matriz de confusão da Figura 5.4, é um bocado prova do que estavamos a dizer acima. Apesar do *dataset* possuir menos imagens de folhas saudáveis (*healthy*) do que das outras duas doenças, o facto de estas se diferenciarem bastante das outras em termos de cor faz com que a percentagem de diagnóstico de folhas saudáveis seja muito alta. Numa das amostras, à qual a matriz de confusão pertence, em 38 imagens de folhas saudáveis, apenas errou em 2 diagnósticos, pois previu que estas teriam podridão negra (*blackrot*). No geral, nesta amostra, em 299 imagens, acertou na classificação de 277 (somatório dos valores na diagonal).
Como podemos observar na Figura 5.5, de entre as 10 amostras testadas, obtemos uma média de 94,11% de acerto.

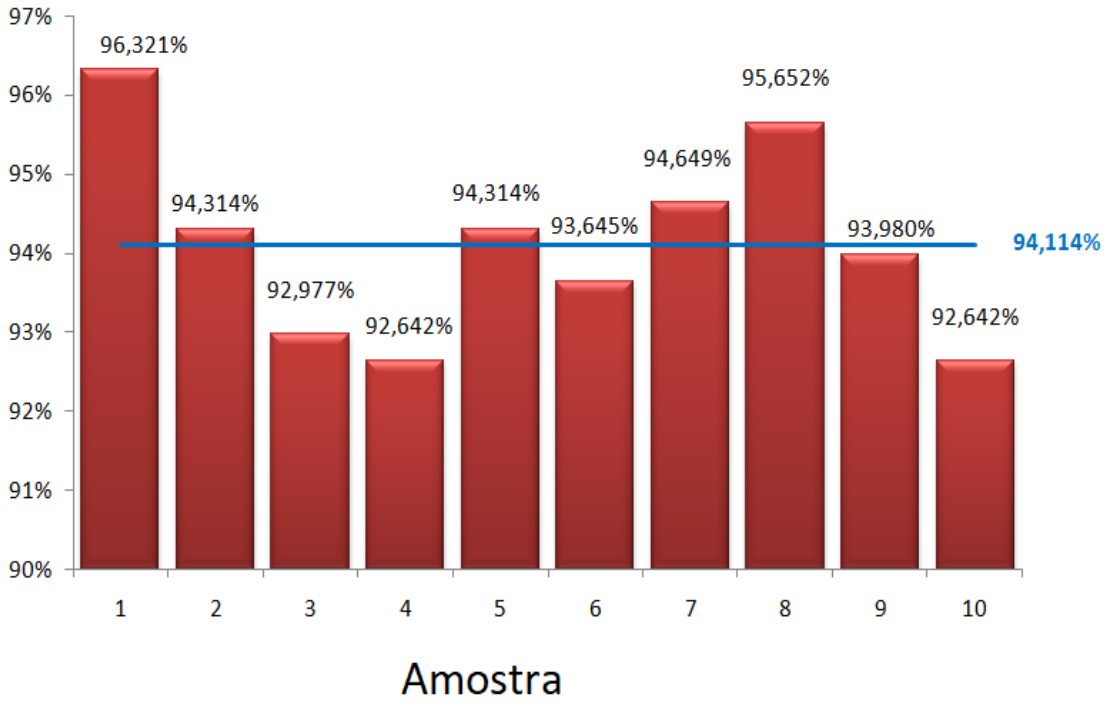

Figura 5.5: Gráfico de Precisão - RGB

5.2.2 Espaço de cores HSV

De seguida passámos à análise da eficácia do histograma HSV como descritor de cor.

		Previsão		
		blackrot	esca	healthy
Real	blackrot	103	6	1
	esca	16	135	0
	healthy	1	0	37

Figura 5.6: Matriz de confusão do descritor - Histograma HSV

A matriz de confusão do histograma HSV para a mesma amostra apresenta melhores resultados na hora de identificar folhas saudáveis, sendo que em 38 apenas errou em 1. Por outro lado, para as outras duas doenças apresenta resultados ligeiramente piores, e no total, em 299 acerta 275. Em comparação com o descritor anterior, nesta amostra em concreto, a precisão diminui, pois errou em mais 2 diagnósticos.
No entanto, no geral, o espaço de cores HSV apresenta resultados ligeiramente superiores aos resultados do espaço de cores RGB. Obtemos um aumento de 0,37% em comparação ao anterior, ficando assim com 94,48% de precisão média para as mesmas 10 amostras, tal como podemos observar na Figura 5.7. Concluímos, então que ambos os descritores apresentam precisões muito altas, sendo que dependendo da amostra de treino, a balança pode tender para uma maior precisão por parte do histograma RGB, enquanto que noutras pode tender para uma maior precisão do histograma HSV.

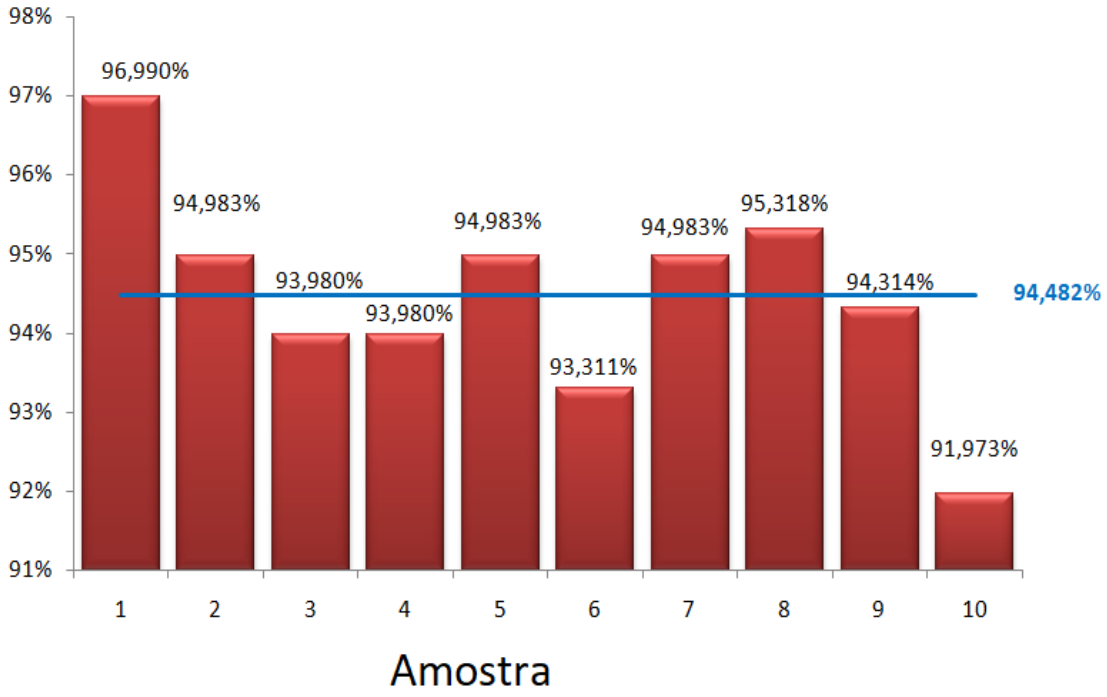

Figura 5.7: Gráfico de Precisão - HSV

5.2.3 LBP

Passámos então para o nosso descritor de textura. Relembrando que implementamos um *Local Binary Pattern*, como único descritor de textura. Esta descrição de textura é feita, tal como foi dito no Capítulo 4, por um histograma com 256 coeficientes, cada um com o nível de tom de cinza da imagem LBP no píxel com o mesmo número.

Começamos então por analisar a matriz de confusão para a mesma amostra usada anteriormente, tendo em conta o histograma LBP como descritor.

Real \ Previsão	blackrot	esca	healthy
blackrot	89	19	2
esca	21	130	0
healthy	0	0	38

Figura 5.8: Matriz de confusão do descritor - Histograma LBP

A matriz de confusão do histograma LBP esta amostra apresenta ainda melhores resultados na hora de identificar folhas saudáveis, sendo que acertou todas as 38. No entanto, os resultados são muito piores na hora de diferenciar a podridão negra da esca. Após exaustiva investigação, chegamos à conclusão que, por o LBP calcular a diferença de tom de cinza de um píxel para os seus píxeis vizinhos, este vai ter valores superiores quantas mais mazelas a folha tenha. Isto deve-se a que as manchas na folha terem um tom de cinza diferente da parte não afectada. Por outro lado, este descritor não vai diferenciar da melhor forma, através da textura, a diferença entre as manchas presentes nos sintomas de uma doença, com os presentes nos sintomas da outra. Das 299 imagens a diagnosticar, o classificador acertou em 257.
Através do gráfico da Figura 5.9, podemos observar uma descida considerável na percentagem de acerto. Obtemos 87,96% de precisão, bastante menos do que com as *features* dos espaços de cor RGB e HSV. Mesmo assim, consideramos um valor bastante alto e confiável, pois irá alertar-nos se existe alguma doença, mesmo que não seja totalmente preciso a especificar qual.

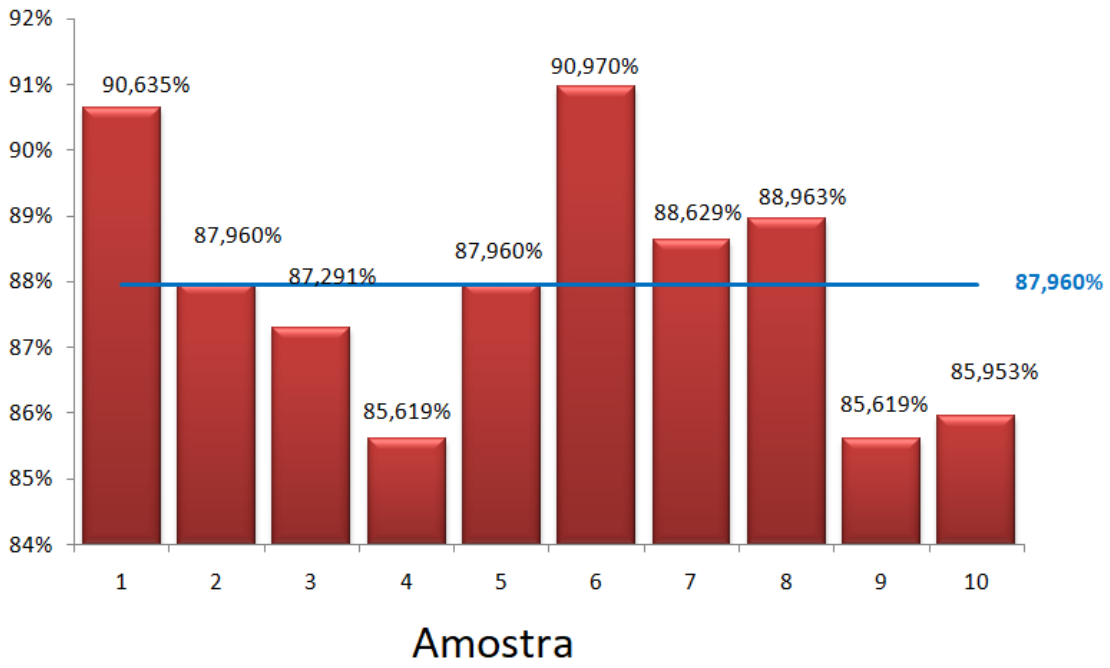

Figura 5.9: Gráfico de Precisão - LBP

5.2.4 Forma

Passámos finalmente para o nosso descritor de forma. Relembrando o que foi dito no Capítulo 4, consiste num histograma com 8 coeficiente, sendo que cada um deles possui a distância em píxeis, em várias direcções, do centro da imagem até à fronteira entre a folha e o fundo. Como já tinhamos previsto, com as *features* de forma, obtemos uma precisão muito inferior, devido ao ruído presente nas imagens que não foi captado nas etapas anteriores do processamento, e que por vezes impossibilita de calcular a distância do centro até à fronteira real. Esta diminuição da precisão também se deve ao facto de, a forma da folha não ser uma característica tão denunciadora do tipo de doença como a cor ou a textura. Como tinhamos visto no Capítulo 2, a doença Podridão Negra é identificável pelas manchas castanho-avermelhadas ou após 3 a 4 dias manchas negras enquanto que a Esca apresenta manchas amareladas ou avermelhadas. Estes são os únicos sintomas, e daí a forma não ser tão evidente na altura de diagnosticar a doença. Podemos observar então, na Figura 5.10, uma grande redução na precisão, para os 57,09%.

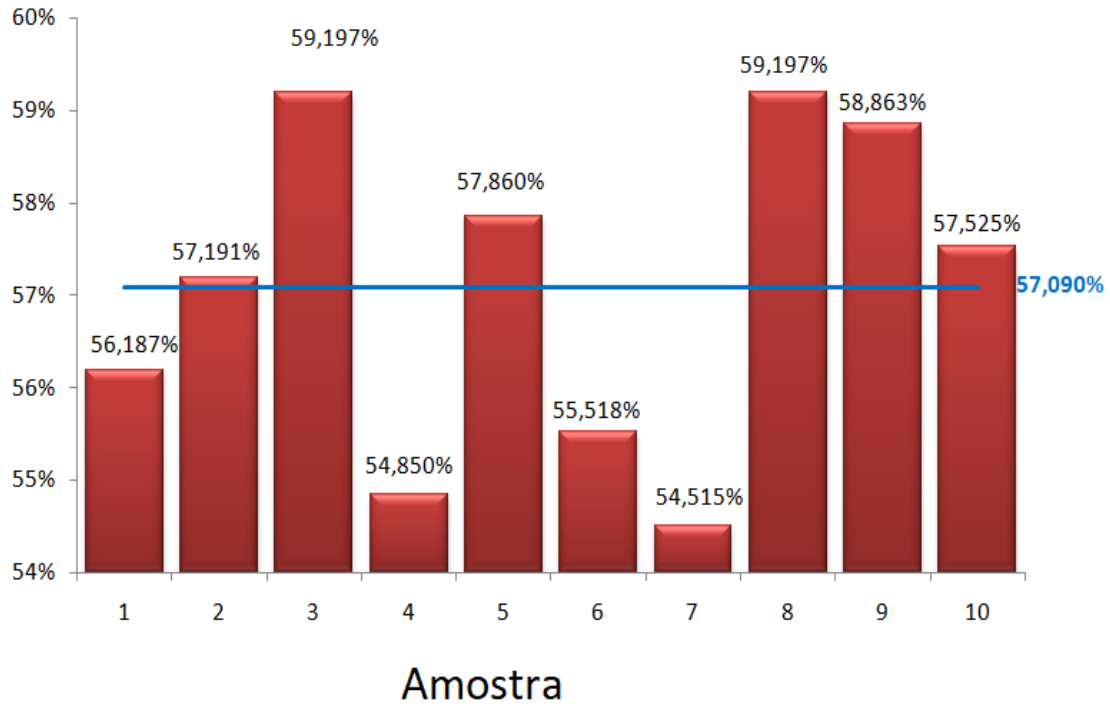

Figura 5.10: Gráfico de Precisão - Forma

No entanto, sendo que temos três classes entre as quais escolher, uma precisão acima dos 50% não é de todo má, e dá para concluir que com o número elevado de imagens do *dataset*, se tenha encontrado uma pequena semelhança de forma, que denuncia, minimamente qual o diagnóstico correcto, se bem que nunca de forma tão eficaz como os outros descritores falados anteriormente.

Real \ Previsão	blackrot	esca	healthy
blackrot	42	67	1
esca	40	111	0
healthy	13	6	19

Figura 5.11: Matriz de confusão do descritor - Histograma Forma

Podemos confirmar esta teoria, através da matriz de confusão para a amostra usada até aqui. Existe um aumento na taxa de erro na hora de diagnosticar uma folha como saudável, assim como para as outras doenças, pois a forma não denuncia a doença de maneira tão evidente. Resultando no total, num acerto de 172 diagnosticos dos 299 possíveis.

5.2.5 Combinações

Consideramos como o seguinte passo, juntar descritores, para testar quais trabalham melhor em conjunto. Após testar várias combinações de *features*, tal como podemos observar nas Figuras 5.12 e 5.13, podemos observar que nem todas as características vão acrescentar precisão ao diagnóstico das doenças.

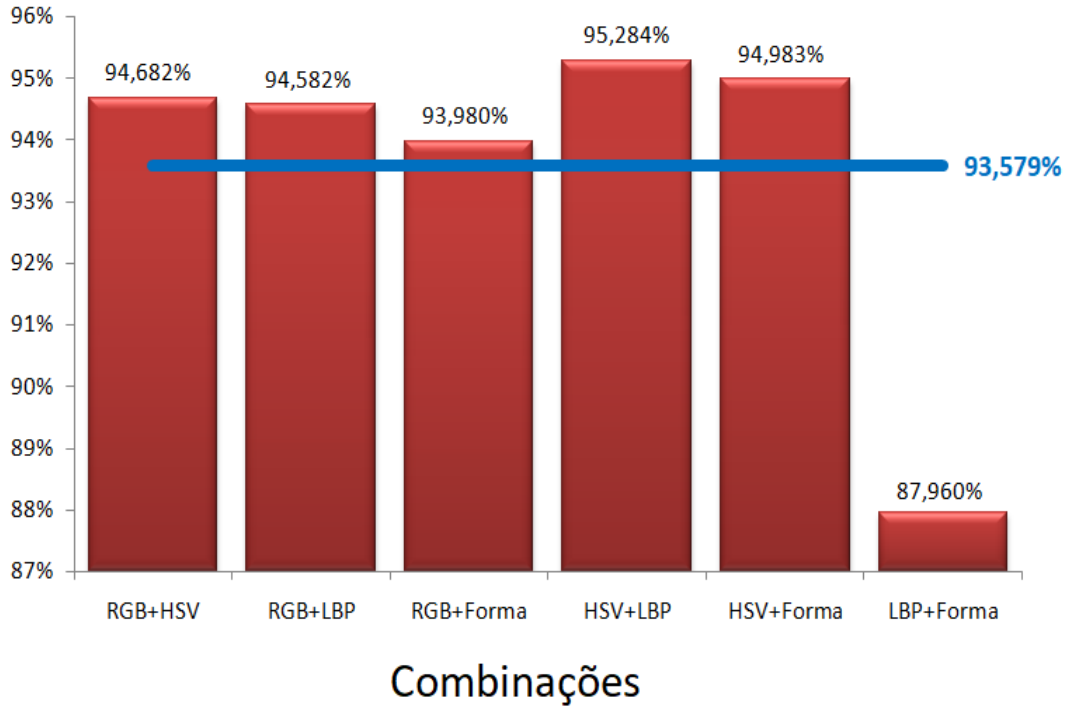

Figura 5.12: Gráfico de Precisão - Combinações de 2

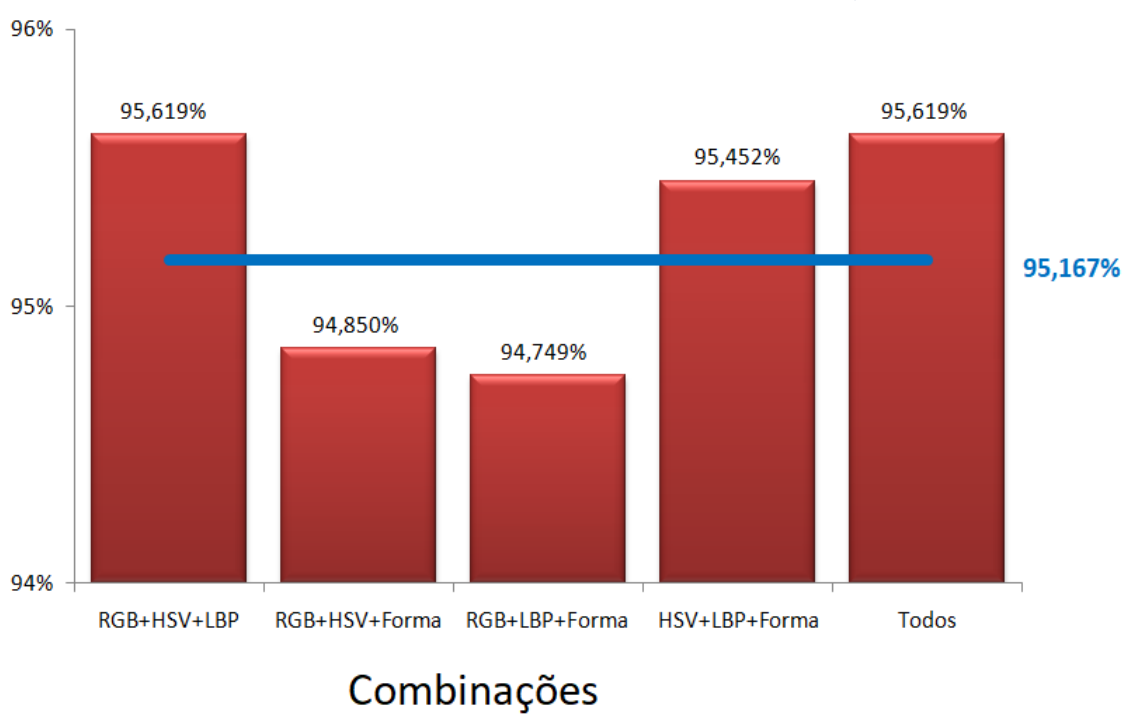

Figura 5.13: Gráfico de Precisão - Combinações de 3 e mais

É possível reparar que não existe diferença na percentagem de acerto ao adicionar as *features* de forma à combinação que melhor percentagem apresenta, que é a combinação do histograma RGB, com histogram HSV e ainda com o histograma LBP.

		Previsão		
		blackrot	esca	healthy
Real	blackrot	102	8	0
	esca	15	136	0
	healthy	0	0	38

Figura 5.14: Matriz de confusão do descritor - RGB + HSV + LBP

Através da matriz de confusão da amostra usada anteriormente, conseguimos verificar que houve um acerto completo no diagnóstico de folhas saudáveis, proveniente da adição das características obtidas com o descritor de textura. Por outro lado, para esta amostra específica, diminuiu o número de acertos no diagnóstico de podridão negra. Quanto ao diagnóstico de esca, o valor manteve-se, em comparação à melhor *feature* para esta doença, que era o histograma HSV.

5.3 Discussão

Após a análise dos vários componentes deste sistema, e do seu funcionamento, conseguímos concluir diversas coisas. As condições de captura das imagens do *dataset* influenciaram muito o desenrolar do trabalho. A presença de reflexos de luz e de sombras, fez com que alguma da informação sobre as características extraídas pelos descritores fossem corrompidas. No entanto, o tamanho do *dataset*, disfarçou um bocado essa lacuna, pois o problema da iluminação está presente na maioria das imagens e torna-se por si só numa característica que está sempre presente. Mesmo com esse problema, conseguimos obter uma precisão bastante alta, principalmente por parte dos descritores de cor. Os histogramas RGB e HSV apresentam respectivamente 94,11% e 94,48%, de precisão média. Ao adicionarmos o as características de textura, por meio do histograma LBP, conseguímos aumentar esta percentagem para 95,17%, sendo este o máximo que conseguímos atingir.

Capítulo 6

Conclusões

Neste capítulo final é resumido o trabalho realizado para esta dissertação e as conclusões a que chegámos. Para além disso, são também apresentadas algumas propostas de funcionalidades com o intuito de melhorar o sistema e investigar mais sobre o tema em questão.

6.1 Trabalho Realizado

Um sistema capaz de diagnosticar a doença presente numa folha de videira através de uma imagem, é uma boa forma de providenciar ao utilizador uma base de decisão para quando aplicar um certo tratamento à videira. Uma ferramenta com uma alta precisão de diagnóstico, que leva a um aumento na probabilidade de cura, pois diferentes doenças necessitam diferentes medicações. Ao acertarmos na identificação da doença reduzimos também a utilização de quimicos, prejudiciais para o ambiente, e que poderão ser desnecessários para a doença em questão.
Fomos capazes de implementar uma ferramenta capaz disso mesmo. Capaz de aproveitar ao máximo a informação presente na imagem, e transformá-la em características associadas à presença ou não de doença, e capaz de identificar a doença em questão. Conseguímos ainda alcançar precisões de diagnóstico muito elevadas, na ordem dos 95%.
Apesar de a precisão ser bastante elevada, temos noção que a inexperiência do autor nesta área de investigação, levou a elevadas perdas de tempo, na hora de implementar certos aspectos do sistema. Este tempo, mal gasto, poderia ter sido melhor aproveitado, para tentar experimentar outros descritores e aplicar outras técnicas de redução de ruído, para fazer com que a precisão aumentasse.
Contudo, além da contribuição científica inerente a esta dissertação, esta proporcionou ao autor boas capacidades, tanto de investigação como de programação, em várias plataformas.

6.2 Trabalho Futuro

Ficará para trabalho a ser desenvolvido no futuro, a incorporação ao *dataset* de imagens de outras doenças, tais como o Mildío e o Oídio, doenças estas mais frequentes em Portugal. Poderá ainda, como mencionado anteriormente, ser feira uma análise e implementação de outros processos de redução de ruído, assim como de outros descritores, com vista a aumentar a precisão de diagnóstico do sistema.
Seria também interessante a melhoria da aplicação *Android*, para proporcionar uma ferramenta prática e comercializável, com informação sobre as doenças, e que medicamentos são recomendados para cada uma.

Apêndice A

Código

A.1 ContrastStretch

```
Mat ContrastStretch(const Mat &src, Mat &dst, float clipHistPercent = 0)
{

    CV_Assert(clipHistPercent >= 0);
    CV_Assert((src.type() == CV_8UC1) || (src.type() == CV_8UC3) ||
        (src.type() == CV_8UC4));

    int histSize = 256;
    float alpha, beta;
    double minGray = 0, maxGray = 0;

    //to calculate grayscale histogram
    Mat gray;
    if (src.type() == CV_8UC1) gray = src;
    else if (src.type() == CV_8UC3) cvtColor(src, gray, COLOR_BGR2GRAY);
    else if (src.type() == CV_8UC4) cvtColor(src, gray, COLOR_BGRA2GRAY);
    if (clipHistPercent == 0)
    {
        // keep full available range
        minMaxLoc(gray, &minGray, &maxGray);
    }
    else
    {
        Mat hist; //the grayscale histogram

        float range[] = { 0, 256 };
        const float* histRange = { range };
        bool uniform = true;
        bool accumulate = false;
        calcHist(&gray, 1, 0, Mat(), hist, 1, &histSize, &histRange, uniform,
            accumulate);

        // calculate cumulative distribution from the histogram
        vector<float> accumulator(histSize);
        accumulator[0] = hist.at<float>(0);
        for (int i = 1; i < histSize; i++)
        {
            accumulator[i] = accumulator[i - 1] + hist.at<float>(i);
        }

        // locate points that cuts at required value
        float max = accumulator.back();
        clipHistPercent *= (max / 100.0); //make percent as absolute
        clipHistPercent /= 2.0; // left and right wings
        // locate left cut
        minGray = 0;
        while (accumulator[minGray] < clipHistPercent)
            minGray++;
```

```
        // locate right cut
        maxGray = histSize - 1;
        while (accumulator[maxGray] >= (max - clipHistPercent))
            maxGray--;
    }

    // current range
    float inputRange = maxGray - minGray;

    alpha = (histSize - 1) / inputRange;    // alpha expands current range to
        histsize range
    beta = -minGray * alpha;                // beta shifts current range so that
        minGray will go to 0

    // Apply brightness and contrast normalization
    // convertTo operates with saurate_cast
    src.convertTo(dst, -1, alpha, beta);

    // restore alpha channel from source
    if (dst.type() == CV_8UC4)
    {
        int from_to[] = { 3, 3 };
        mixChannels(&src, 4, &dst, 1, from_to, 1);
    }
    return dst;
}
```

A.2 saturationEnhancement

```
Mat saturationEnhancement(Mat image) {

   Mat hsv_image;

   cvtColor(image, hsv_image, COLOR_RGB2HSV);

   int hue = 0;
   int saturation = 0;
   int value = 0;

   for (int y = 0; y < hsv_image.rows; y++) {
      for (int x = 0; x < hsv_image.cols; x++) {
         //Hue
         hue = image.at<Vec3b>(y, x)[0];
         //Saturation
         saturation = image.at<Vec3b>(y, x)[1];
         //Value
         value = image.at<Vec3b>(y, x)[2];

         //image.at<Vec3b>(y, x)[2] = image.at<Vec3b>(y, x)[2] + 20;
         hsv_image.at<Vec3b>(y, x)[1] = 200;


      }
   }
   cvtColor(hsv_image, image, COLOR_HSV2RGB);

   return image;

}
```

A.3 mantainGreen

```
Mat mantainGreen(Mat image) {
   int blue = 0;
   int green = 0;
   int red = 0;

   for (int y = 0; y < image.rows; y++) {
      for (int x = 0; x < image.cols; x++) {
         //Blue
         blue = image.at<Vec3b>(y, x)[0];
         //Green
         green = image.at<Vec3b>(y, x)[1];
         //Red
         red = image.at<Vec3b>(y, x)[2];

         if (green < blue || green < red || (green > 60 && red > 60 && blue >
             60)) {
            image.at<Vec3b>(y, x) = 0;
         }
      }
   }
   return image;
}
```

A.4 mergeGreenWithOriginal

```
Mat mergeGreenWithOriginal(Mat original, Mat green) {
   Mat new_image = green.clone();
   for (int y = 0; y < green.rows; y++) {
      for (int x = 0; x < green.cols; x++) {
         if (green.at<Vec3b>(y, x)[0] == 0 && green.at<Vec3b>(y, x)[1] == 0
             && green.at<Vec3b>(y, x)[2] == 0) {
            new_image.at<Vec3b>(y, x)[0] = original.at<Vec3b>(y,x)[0];
            new_image.at<Vec3b>(y, x)[1] = original.at<Vec3b>(y, x)[1];
            new_image.at<Vec3b>(y, x)[2] = original.at<Vec3b>(y, x)[2];
         }
      }
   }
   return new_image;
}
```

A.5 mergeTwo

```
Mat mergeTwo(Mat imageBack, Mat imageShadow) {

    Mat new_image = imageBack.clone();
    for (int y = 0; y < imageBack.rows; y++) {
        for (int x = 0; x < imageBack.cols; x++) {
            if (imageShadow.at<Vec3b>(y, x)[0] == 0 && imageShadow.at<Vec3b>(y,
                x)[1] == 0 && imageShadow.at<Vec3b>(y, x)[2] == 0) {
                new_image.at<Vec3b>(y, x)[0] = 0;
                new_image.at<Vec3b>(y, x)[1] = 0;
                new_image.at<Vec3b>(y, x)[2] = 0;
            }
        }
    }
    return new_image;
}
```

A.6 removeBack

```
Mat removeBack(Mat image) {
    int blue = 0;
    int green = 0;
    int red = 0;

    for (int y = 0; y < image.rows; y++) {
        for (int x = 0; x < image.cols; x++) {
            //Blue
            blue = image.at<Vec3b>(y, x)[0];
            //Green
            green = image.at<Vec3b>(y, x)[1];
            //Red
            red = image.at<Vec3b>(y, x)[2];

            if (green > 50 && blue > 50 && red > 50){
                image.at<Vec3b>(y, x) = {0,0,0};
            }
        }
    }
    return image;
}
```

A.7 binaryImage

```
Mat binaryImage(Mat image) {
   int blue = 0;
   int green = 0;
   int red = 0;

   for (int y = 0; y < image.rows; y++) {
      for (int x = 0; x < image.cols; x++) {
         //Blue
         blue = image.at<Vec3b>(y, x)[0];
         //Green
         green = image.at<Vec3b>(y, x)[1];
         //Red
         red = image.at<Vec3b>(y, x)[2];

         if (green == 0 && blue == 0 && red == 0) {
            image.at<Vec3b>(y, x) = { 0,0,0 };
         }
         else image.at<Vec3b>(y, x) = { 255,255,255 };
      }
   }
   return image;
}
```

A.8 RGBHistogram

```
int get5Value(int val) {

   if (val == 0) {
      return 8;
   }
   if (val >= 1 && val < 52) {
      return 0;
   }
   if (val >= 52 && val < 103) {
      return 1;
   }
   if (val >= 103 && val < 154) {
      return 2;
   }
   if (val >= 154 && val < 205) {
      return 3;
   }
   if (val >= 205 && val < 256) {
      return 4;
   }
}


String RGBHistogram(Mat image, String name) {

   vector<vector<vector<int>>> rgb_val(5, vector<vector<int>>(5,
       vector<int>(5)));

   name = name.substr(0, name.size() - 4);

   int a;
   int b;
   int c;
   for (int i = 0; i < image.rows; i++) {
      for (int j = 0; j < image.cols; j++) {
         a = get5Value(image.at<Vec3b>(j,i)[0]);
         b = get5Value(image.at<Vec3b>(j, i)[1]);
         c = get5Value(image.at<Vec3b>(j, i)[2]);
         if (a != 8 && b != 8 && c != 8) {
            rgb_val[a][b][c]++;
         }
         else {
            if (a == 8 && b == 8 && c != 8) {
               rgb_val[0][0][c]++;
            }
            if (a == 8 && b != 8 && c == 8) {
               rgb_val[0][b][0]++;
            }
```

```
            if (a != 8 && b == 8 && c == 8) {
               rgb_val[a][0][0]++;
            }

            if (a == 8 && b != 8 && c != 8) {
               rgb_val[0][b][c]++;
            }
            if (a != 8 && b == 8 && c != 8) {
               rgb_val[a][0][c]++;
            }
            if (a != 8 && b != 8 && c == 8) {
               rgb_val[a][b][0]++;
            }

         }

      }
   }

   String hist = "";
   for (int i = 0; i < rgb_val.size(); i++) {
      for (int j = 0; j < rgb_val[i].size(); j++) {
         for (int k = 0; k < rgb_val[i][j].size(); k++) {
            stringstream ss;
            ss << rgb_val[i][j].at(k);
            string s = ss.str();
            hist.append(s + ";");
         }
      }
   }

   hist = hist.substr(0, hist.size() - 1);
   return hist;

}
```

A.9 HSVHistogram

```
int get16val(int val) {

   if (val >= 0 && val < 16) {
      return 0;
   }
   if (val >= 16 && val < 32) {
      return 1;
   }
   if (val >= 32 && val < 48) {
      return 2;
   }
   if (val >= 48 && val < 64) {
      return 3;
   }
   if (val >= 64 && val < 80) {
      return 4;
   }
   if (val >= 80 && val < 96) {
      return 5;
   }
   if (val >= 96 && val < 112) {
      return 6;
   }
   if (val >= 112 && val < 128) {
      return 7;
   }
   if (val >= 128 && val < 144) {
      return 8;
   }
   if (val >= 144 && val < 160) {
      return 9;
   }
   if (val >= 160 && val < 176) {
      return 10;
   }
   if (val >= 176 && val < 192) {
      return 11;
   }
   if (val >= 192 && val < 208) {
      return 12;
   }
   if (val >= 208 && val < 224) {
      return 13;
   }
   if (val >= 224 && val < 240) {
      return 14;
   }
   if (val >= 240 && val < 256) {
```

```
        return 15;
    }

}

int get4val(int val) {

    if (val >= 1 && val < 64) {
        return 0;
    }
    if (val >= 64 && val < 128) {
        return 1;
    }
    if (val >= 128 && val < 192) {
        return 2;
    }
    if (val >= 192 && val < 256) {
        return 3;
    }

}

int get4val0(int val) {

    if (val == 0) {
        return 4;
    }

    if (val >= 0 && val < 64) {
        return 0;
    }
    if (val >= 64 && val < 128) {
        return 1;
    }
    if (val >= 128 && val < 192) {
        return 2;
    }
    if (val >= 192 && val < 256) {
        return 3;
    }

}

String HSVHistogram(Mat image, String name) {

    Mat hsv_image;
    cvtColor(image, hsv_image, COLOR_RGB2HSV);

    vector<vector<vector<int>>> hsv_val(16, vector<vector<int>>(4,
        vector<int>(4)));
```

```
    int a;
    int b;
    int c;
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            a = get16val(image.at<Vec3b>(j, i)[0]);
            b = get4val(image.at<Vec3b>(j, i)[1]);
            c = get4val0(image.at<Vec3b>(j, i)[2]);

            if (c != 4) {
                hsv_val[a][b][c]++;
            }
        }
    }


    String hist = "";
    for (int i = 0; i < hsv_val.size(); i++) {
        for (int j = 0; j < hsv_val[i].size(); j++) {
            for (int k = 0; k < hsv_val[i][j].size(); k++) {
                stringstream ss;
                ss << hsv_val[i][j].at(k);
                string s = ss.str();
                hist.append(s + ";");
            }
        }
    }


    hist = hist.substr(0, hist.size() - 1);

    return hist;

}
```

A.10 LBP

```
Mat LBP(Mat image) {
   bool affiche = true;
   vector<Mat> textureData;
   Mat lbp(image.rows, image.cols, CV_8UC1);
   Mat Image;
   cvtColor(image, Image, COLOR_BGR2GRAY);
   unsigned center = 0, center_lbp = 0;
   Mat H;
   for (int row = 1; row < Image.rows - 1; row++)
   {
      for (int col = 1; col < Image.cols - 1; col++)
      {
         center = Image.at<uchar>(row, col);
         center_lbp = 0;
         if (center <= Image.at<uchar>(row - 1, col - 1))
            center_lbp += 1;
         if (center <= Image.at<uchar>(row - 1, col))
            center_lbp += 2;
         if (center <= Image.at<uchar>(row - 1, col + 1))
            center_lbp += 4;
         if (center <= Image.at<uchar>(row, col - 1))
            center_lbp += 128;
         if (center <= Image.at<uchar>(row, col + 1))
            center_lbp += 8;
         if (center <= Image.at<uchar>(row + 1, col - 1))
            center_lbp += 64;
         if (center <= Image.at<uchar>(row + 1, col))
            center_lbp += 32;
         if (center <= Image.at<uchar>(row + 1, col + 1))
            center_lbp += 16;
         lbp.at<uchar>(row, col) = center_lbp;
      }
   }
   return lbp;
}
```

A.11 get_256_Histogram

```
vector<int> get_256_Histogram(Mat image) {
    vector<int> values(256, 0);
    for (int y = 0; y < image.rows; y++) {
        for (int x = 0; x < image.cols; x++) {
            int val = image.at<uchar>(y, x);
            values[val]++;
        }
    }
    return values;
}
```

A.12 checkBorder

```
String checkBorder(Mat image, String name) {
    /*
    1 - right
    2 - RU
    3 - up
    4 - LU
    5 - left
    6 - LD
    7 - down
    8 - RD

    */

    Mat imageclone = image.clone();
    String result = "";
    int distR = 0;
    int distRU = 0;
    int distU = 0;
    int distLU = 0;
    int distL = 0;
    int distLD = 0;
    int distD = 0;
    int distRD = 0;
    int x = 127;
    int y = 127;
    //Right
    for (int i = 0; i < 127; i++) {
        if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
            image.at<Vec3b>(y, x)[2] != 0) {
            distR++;
            imageclone.at<Vec3b>(y, x) = (0,255,0);
            x++;
```

```
    }
    else {
        ostringstream ss;
        ss << distR;
        String distRight = ss.str();
        result.append(distRight);
        result.append(";");
        break;

    }
    if (i == 127) {
        distR = 127;
        ostringstream ss;
        ss << distR;
        String distRight = ss.str();
        result.append(distRight);
        result.append(";");
        break;
    }
}

//Right Up
x = 127;
y = 127;
for (int i = 0; i < 127; i++) {
    if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
        image.at<Vec3b>(y, x)[2] != 0) {
        distRU++;
        imageclone.at<Vec3b>(y, x) = (0, 255, 0);
        x++;
        y++;
    }
    else {
        ostringstream ss;
        ss << distRU;
        String distRightUp = ss.str();
        result.append(distRightUp);
        result.append(";");
        break;

    }
    if (i == 127) {
        distR = 127;
        ostringstream ss;
        ss << distR;
        String distRight = ss.str();
        result.append(distRight);
        result.append(";");
        break;
    }
}
```

```
//Up
x = 127;
y = 127;
for (int i = 0; i < 127; i++) {
   if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
       image.at<Vec3b>(y, x)[2] != 0) {
      distU++;
      imageclone.at<Vec3b>(y, x) = (0, 255, 0);
      y++;
   }
   else {
      ostringstream ss;
      ss << distU;
      String distUp = ss.str();
      result.append(distUp);
      result.append(";");
      break;

   }
   if (i == 127) {
      distR = 127;
      ostringstream ss;
      ss << distR;
      String distRight = ss.str();
      result.append(distRight);
      result.append(";");
      break;
   }
}

//Left Up
x = 127;
y = 127;
for (int i = 0; i < 127; i++) {
   if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
       image.at<Vec3b>(y, x)[2] != 0) {
      distLU++;
      imageclone.at<Vec3b>(y, x) = (0, 255, 0);
      x--;
      y++;
   }
   else {
      ostringstream ss;
      ss << distLU;
      String distLeftUp = ss.str();
      result.append(distLeftUp);
      result.append(";");
      break;

   }
```

```
    if (i == 127) {
        distR = 127;
        ostringstream ss;
        ss << distR;
        String distRight = ss.str();
        result.append(distRight);
        result.append(";");
        break;
    }
}

//Left
x = 127;
y = 127;
for (int i = 0; i < 127; i++) {
    if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
        image.at<Vec3b>(y, x)[2] != 0) {
        distL++;
        imageclone.at<Vec3b>(y, x) = (0, 255, 0);
        x--;

    }
    else {
        ostringstream ss;
        ss << distL;
        String distLeft = ss.str();
        result.append(distLeft);
        result.append(";");
        break;

    }
    if (i == 127) {
        distR = 127;
        ostringstream ss;
        ss << distR;
        String distRight = ss.str();
        result.append(distRight);
        result.append(";");
        break;
    }
}

//Left Down
x = 127;
y = 127;
for (int i = 0; i < 127; i++) {
    if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
        image.at<Vec3b>(y, x)[2] != 0) {
        distLD++;
        imageclone.at<Vec3b>(y, x) = (0, 255, 0);
        x--;
```

```
            y--;
        }
        else {
            ostringstream ss;
            ss << distLD;
            String distLeftDown = ss.str();
            result.append(distLeftDown);
            result.append(";");
            break;

        }
        if (i == 127) {
            distR = 127;
            ostringstream ss;
            ss << distR;
            String distRight = ss.str();
            result.append(distRight);
            result.append(";");
            break;
        }
    }

    //Down
    x = 127;
    y = 127;
    for (int i = 0; i < 127; i++) {
        if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
            image.at<Vec3b>(y, x)[2] != 0) {
            distD++;
            imageclone.at<Vec3b>(y, x) = (0, 255, 0);
            y--;
        }
        else {
            ostringstream ss;
            ss << distD;
            String distDown = ss.str();
            result.append(distDown);
            result.append(";");
            break;

        }
        if (i == 127) {
            distR = 127;
            ostringstream ss;
            ss << distR;
            String distRight = ss.str();
            result.append(distRight);
            result.append(";");
            break;
        }
    }
```

```
    //Right Down
    x = 127;
    y = 127;
    for (int i = 0; i < 127; i++) {
        if (image.at<Vec3b>(y, x)[0] != 0 && image.at<Vec3b>(y, x)[1] != 0 &&
            image.at<Vec3b>(y, x)[2] != 0) {
            distRD++;
            imageclone.at<Vec3b>(y, x) = (0, 255, 0);
            x++;
            y--;
        }
        else {
            ostringstream ss;
            ss << distRD;
            String distRightDown = ss.str();
            result.append(distRightDown);
            break;

        }
        if (i == 127) {
            distR = 127;
            ostringstream ss;
            ss << distR;
            String distRight = ss.str();
            result.append(distRight);
            result.append(";");
            break;
        }
    }

    return result;
}
```

A.13 Aplicação Android

```
package com.example.pc.teste;

import android.Manifest;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.net.Uri;
import android.provider.MediaStore;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import net.gotev.uploadservice.MultipartUploadRequest;
import net.gotev.uploadservice.ServerResponse;
import net.gotev.uploadservice.UploadInfo;
import net.gotev.uploadservice.UploadNotificationConfig;
import net.gotev.uploadservice.UploadStatusDelegate;
import java.io.IOException;
import java.util.UUID;

public class MainActivity extends AppCompatActivity implements
    View.OnClickListener {
    private Button select, send, disease;
    private ImageView imageview;
    private TextView response;
    //private EditText editText;
    private static final int STORAGE_PERMISSION_CODE = 2342;
    private static final int PICK_IMAGE_REQUEST = 22;
    private Uri filePath;
    private Bitmap bitmap;


    private static final String UPLOAD_URL =
        "http://192.168.56.1/UploadExample/Project1/x64/Debug/upload.php";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        requestStoragePermission();
        select = (Button) findViewById(R.id.b_select);
        send = (Button) findViewById(R.id.b_send);
        //disease = (Button) findViewById(R.id.b_disease);
        imageview = (ImageView) findViewById(R.id.image);
        //editText = (EditText) findViewById(R.id.editText);
        response = (TextView) findViewById(R.id.response);
        select.setOnClickListener(this);
        send.setOnClickListener(this);


    }
    private void requestStoragePermission() {
        if(ContextCompat.checkSelfPermission(this,Manifest.permission.
        READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED) {
            return;
        }
        ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
            STORAGE_PERMISSION_CODE);

    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
        permissions, @NonNull int[] grantResults) {
        if(requestCode == STORAGE_PERMISSION_CODE) {
            if(grantResults.length > 0 && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission Granted",
                    Toast.LENGTH_LONG).show();
            }
            else {
                Toast.makeText(this, "Permission Not Granted",
                    Toast.LENGTH_LONG).show();
            }
        }
    }

    private void showFileChooser(){
        Intent intent = new Intent();
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
        startActivityForResult(Intent.createChooser(intent,"Select Picture"),
            PICK_IMAGE_REQUEST);
    }

    @Override
```

```
protected void onActivityResult(int requestCode, int resultCode,
    @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if(requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK &&
        data != null && data.getData() != null){
        filePath = data.getData();
        try{
            bitmap =
                MediaStore.Images.Media.getBitmap(getContentResolver(),
            filePath);
            imageview.setImageBitmap(bitmap);
        }
        catch (IOException e){

        }
    }
}

private String getPath(Uri uri){
    Cursor cursor = getContentResolver().query(uri,null,null,null,null);
    cursor.moveToFirst();
    String document_id = cursor.getString(0);

    document_id = document_id.substring(document_id.lastIndexOf(":")+1);
    cursor.close();

    cursor = getContentResolver().query(
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
            null, MediaStore.Images.Media._ID + " = ? ", new
                String[]{document_id}, null
    );
    cursor.moveToFirst();
    String path =
        cursor.getString(cursor.getColumnIndex(MediaStore.Images.Media.
    DATA));
    cursor.close();
    return path;

}


private String uploadImage(){

    String name = "simon";
            //editText.getText().toString().trim();
    String path = getPath(filePath);
    String uploadid = "";

    try{
```

```
            uploadid = UUID.randomUUID().toString();
            String text = new MultipartUploadRequest(this,uploadid,UPLOAD_URL)
                .addFileToUpload(path, "image")
                .addParameter("name", name)
                .setNotificationConfig(new UploadNotificationConfig())
                .setMaxRetries(2)
                .setDelegate(new UploadStatusDelegate() {
                    private String res;
                    @Override
                    public /*synchronized*/ void onProgress(Context context,
                        UploadInfo uploadInfo) {
                        Log.e("gotev","onProgress
                            eltstr="+uploadInfo.getElapsedTimeString()+"
                            elt="+uploadInfo.getElapsedTime()+" ratestr="+
                                uploadInfo.getUploadRateString()+"
                                    prc="+uploadInfo.getProgressPercent()+"
                                    tb="+uploadInfo.getTotalBytes()+"
                                    ub="+uploadInfo.getUploadedBytes());

                    }
                    @Override
                    public /*synchronized*/ void onError(Context context,
                        UploadInfo uploadInfo, Exception exception) {
                        //Log.e("gotev","onError "+e.getMessage() +
                            e.getStackTrace());

                    }
                    @Override
                    public /*synchronized*/ void onCompleted(Context context,
                        UploadInfo uploadInfo, ServerResponse serverResponse)
                        {
                        Log.e("gotev","onCompleted
                            "+serverResponse.getHttpCode()+"
                            "+serverResponse.getBodyAsString());
                        res = serverResponse.getBodyAsString();
                        res = res.substring(1,res.length()-1);
                        res = res.toUpperCase();
                        if(res.equals("BLACKROT")){
                            response.setText("Foi diagnosticada a doen a
                                PODRIDO NEGRA!");
                        }
                        else{
                            if(res.equals("ESCA")){
                                response.setText("Foi diagnosticada a doen a
                                    ESCA!");
                            }
                            else {
                                response.setText("A folha analisada
                                    encontra-se SAUDVEL!");
                            }
                        }
```

```
                    }
                    @Override
                    public /*synchronized*/ void onCancelled(Context context,
                        UploadInfo uploadInfo) {
                        Log.e("gotev","onCancelled "+uploadInfo);

                    }

                }).startUpload();


            //Toast.makeText(this, text, Toast.LENGTH_LONG).show();
            return uploadid;
        }
        catch (Exception e){
            Toast.makeText(this, "Check", Toast.LENGTH_LONG).show();
        }

        return uploadid;
    }

    @Override
    public void onClick(View view) {
        String id = "";
        String dis = "";
        if(view == select){
            showFileChooser();
        }
        if(view == send){
            id = uploadImage();
        }
       /* if(view == disease){
            getDisease(id);
        }
        */

    }
}
```

A.14 dice

```
float dice(Mat image1, Mat image2)
{
   Mat n1 = image1.clone();
   Mat n2 = image2.clone();
   float intersect = 0;
   float reunion = 0;

   float dice = 0;
   for (int i = 0; i < n1.rows; i++) {
      for (int j = 0; j < n1.cols; j++) {
         if ((n1.at<Vec3b>(i, j)[0] == 255 && n1.at<Vec3b>(i, j)[1] == 255 &&
             n1.at<Vec3b>(i, j)[2] == 255) || (n2.at<Vec3b>(i, j)[0] == 255
             && n2.at<Vec3b>(i, j)[1] == 255 && n2.at<Vec3b>(i, j)[2] ==
             255)) {
            reunion++;
            if ((n1.at<Vec3b>(i, j)[0] == 255 && n1.at<Vec3b>(i, j)[1] == 255
                && n1.at<Vec3b>(i, j)[2] == 255) && (n2.at<Vec3b>(i, j)[0] ==
                255 && n2.at<Vec3b>(i, j)[1] == 255 && n2.at<Vec3b>(i, j)[2]
                == 255)) {
               intersect++;
            }
         }
      }
   }

   dice = intersect/reunion;
   return dice;
}
```

Apêndice B

Filtragem dos Artigos

Titulo	Método de Filtragem			
	Por Título	Por *Abstract*	Por leitura	
2019				
Protocol for the Definition of a Multi-Spectral Sensor for Specific Foliar Disease Detection: Case of "Flavescence Dorée"				
2018				
Application of image recognition technology in agricultural production process				
Computer Vision and Machine Learning for Viticulture Technology				
A leaf-wall-to-spray-device distance and leaf-wall-density-based automatic route-planning spray algorithm for vineyards				
2017				
Plant disease leaf image segmentation based on superpixel clustering and EM algorithm				
A Crop Disease Image Retrieval Method Based on the Improvement of Inverted Index				
Preliminary research on the identification system for anthracnose and powdery mildew of sandalwood leaf based on image processing				
Recognition of multiple plant leaf diseases based on improved convolutional neural network				
Performance analysis of combined descriptors for similar crop disease image retrieval				
Mobile Application for Plant Disease Classification Based on Symptom Signatures				
A smart phone image processing application for plant disease diagnosis				
UAS, sensors, and data processing in agroforestry: a review towards practical applications				
Research on Segmentation of Grape Disease Based on Improved Watershed Algorithm				
An individual grape leaf disease identification using leaf skeletons and KNN classification				✓
Detection, categorization and suggestion to cure infected plants of tomato and grapes by using OpenCV framework for andriod environment				✓
Vision Based Classification of Different Diseases of Grape Leaves and their Severity				✓
Nanostructured lipid carriers (NLC) for the delivery of natural molecules with antimicrobial activity: production, characterisation and in vitro studies				
Research on recognition of Empoasca flavescens based on machine vision				
Multispectral band selection for imaging sensor design for vineyard disease detection: case of Flavescence Dorée				
Grape leaf disease detection and classification using multi-class support vector machine				✓
2016				
Image Feature Extraction and Recognition of Plant Leaf Disease in Complex Background				
Detection and classification of diseases of Grape plant using opposite colour Local Binary Pattern feature and machine learning for automated Decision Support System				✓
Random forest based classification of diseases in grapes from images captured in uncontrolled environment				✓
Identification of alfalfa leaf diseases using image recognition technology				
Early detection of grapes diseases using machine learning and IoT				
SVM classifier based grape leaf disease detection				✓
Fusion classification technique used to detect downy and Powdery Mildew grape leaf diseases				✓
Remote hyperspectral imaging of grapevine leafroll-associated virus 3 in cabernet sauvignon vineyards				
Grape Leaf Disease Detection using Embedded Processor				✓
An Unique Technique for Grape Leaf Disease Detection				
Diagnostic Model for Wheat Leaf Conditions Using Image Features and a Support Vector Machine				
2015				
Image Recognition of Maize Leaf Disease Based on GA-SVM				
Research on tobacco disease auto-recognition based on image processing and fuzzy recognition technology				
Recent studies of image and soft computing techniques for plant disease recognition and classification				
Review of the Development of Facilities Vegetable Diseases Diagnosis Technology				
Study on Apple Leaf Disease Recognition Method Based on Support Vector Machine				
A new method for soybean leaf disease detection based on modified salient regions				
2014				
Recognition of cucumber diseases based on leaf image and environmental information				
Tree leaves identification system based on Android mobile phone				
Image-based analysis to study plant infection with human pathogens				
Grape leaf diseases detection & analysis using SGDM matrix method				
DETECTION & DIAGNOSIS OF PLANT LEAF DISEASE USING INTEGRATED IMAGE PROCESSING APPROACH				
Feature extraction and classification method of multi-pose pests using machine vision				
GIS, mechanistic modelling and ontology: a performing mix for precision and sustainable viticulture				
Severity identification of Potato Late Blight disease from crop images captured under uncontrolled environment				
2013				
IMAGE SEGMENTATION METHOD FOR COTTON MITE DISEASE BASED ON COLOR FEATURES AND AREA THRESHOLDING				
IMAGE SEGMENTATION METHOD FOR COTTON MITE DISEASE IMAGE BASED ON COLOUR AND SHAPE FEATURES				
Research on identification method of apple disease based on gray relation analysis				
and Pso-hp Neural Network				
Color image segmentation of plant lesion using improved CV model based on Gaussian distribution				
Precision Agriculture' 13				
A New Clustering Method by Combining Semi-supervised Algorithm and SVM				
Diagnosis and classification of grape leaf diseases using neural networks				✓
Automatic Identification of Crop Diseases Image Based on the Dual Coding Genetic Algorithm of Support Vector Machine [J]				
Recognition of corn leaf disease based on quantum neural network and combination characteristic parameter				
2012				
Study surveys on image segmentation of plant disease spot				
Image recognition of navel orange diseases and insect pests based on compensatory fuzzy neural networks				
Review of Crop Diseases Recognition Based on Image Processing [J]				
Image recognition of plant diseases based on backpropagation networks				
Application of neural networks to image recognition of plant diseases				
Image recognition of plant diseases based on principal component analysis and neural networks				
An improved KPCA/GA-SVM classification model for plant leaf disease recognition				
A Novel Semi Supervised Fuzzy Clustering Algorithm Based on Mahalanobis Distance [J]				
Research on Disease Level of Cucumber Downy Mildew Based on Partial Fuzzy C-means Algorithm [J]				
Image Recognition of Wheat Diseases Based on Improved Color Feature [J]				
Fault diagnosis of diesel engine based on energy and time frequency domain characteristics of vibration signals IMF				
Research on Pattern Recognition Diagnosis of Tilletia Diseases [J]				
Multi-instance graph approach to wheat leaf disease segmentation				
Application of Neural Network to the Prediction of the Number of Domestic Tourists				
Study on diagnosis of Tilletia based on image recognition [J]				
Purity identification of maize seed based on discrete wavelet transform and BP neural network				
Detection and measurement of plant disease symptoms using visible-wavelength photography and image analysis				
A comparative study in kernel-based support vector machine of oil palm leaves nutrient disease				
Ano desconhecido				
Segmentation of Corn Leaf Disease Based on Fully Convolution Neural Network				
Survey on Problem of Early Disease Detection and Monitoring Large Filed Of Crop				
A Windows Phone Application for Plant Disease Diagnosis				
GRAPE LEAF DISEASES DETECTION & ANALYSIS USING CCM METHOD				
NOVEL ALGORITHM FOR GRAPE LEAF DISEASES DETECTION				

Figura B.1: Metodologia de Filtragem de Artigos

Bibliografia

[1] Exportação/expedição de vinhos em agosto 2018. , Instituto do Vinho e da Vinha, 2018 [visited 2018-11-21].

[2] Madalena Neves. Pragas e doenças da vinha. *Direcção Regional de Agricultura da Beira Litoral*, 2000.

[3] Nitesh Agrawal, Jyoti Singhai, and Dheeraj K. Agarwal. Grape leaf disease detection and classification using multi-class support vector machine. *Proceeding International conference on Recent Innovations is Signal Processing and Embedded Systems (RISE-2017)*, 2017. doi:10.1109/RISE.2017.8378160.

[4] Patrice This, Thierry Lacombe, and Mark R.Thomas. Historical origins and genetic diversity of wine grapes. *Trends in genetics*, 2006. doi:10.1016/j.tig.2006.07.008.

[5] Sean Myles, Adam R. Boyko, Christopher L. Owens, Patrick J. Brown, Fabrizio Grassi, Mallikarjuna K. Aradhya, Bernard Prins, Andy Reynolds, Jer-Ming Chia, Doreen Ware, Carlos D. Bustamante, and Edward S. Buckler. Genetic structure and domestication history of the grape. *National Academy of Sciences*, 2010. doi:10.1073/pnas.1009363108.

[6] Informação - podridão negra da videira – "black rot". , Ministério da Agricultura, Mar, Ambiente e Ordenamento de Território [visited 2018-06-19].

[7] Esca na videira. , Ministério da Agricultura, do Desenvolvimento Rural e das Pescas, 2006 [visited 2018-06-19].

[8] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2010.

[9] David A. Forsyth and Jean Ponce. *Computer Vision: A modern approach*. 2012.

[10] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (2nd Edition)*. 2002.

[11] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. 1999.

[12] Ching-Chung Yang. Image enhancement by modified contrast-stretching manipulation. *Optics Laser Technology (OPT LASER TECHNOL)*, 2004. doi:10.1016/j.optlastec.2004.11.009.

[13] S. Arora, J. Acharya, A. Verma, and Prasanta K. Panigrahi. Multilevel thresholding for image segmentation through a fast statistical recursive algorithm. *Pattern Recognition Letters*, 2007. doi:10.1016/j.patrec.2007.09.005.

[14] Gaurav Kumar and Pradeep Kumar Bhatia. A detailed review of feature extraction in image processing systems. *2014 Fourth International Conference on Advanced Computing Communication Technologies*, 2014. doi:10.1109/ACCT.2014.74.

[15] Shamik Sural, Gang Qian, and Sakti Pramanik. Segmentation and histogram generation using the hsv color space for image retrieval. *International Conference on Image Processing*, 2002. doi:10.1109/ICIP.2002.1040019.

[16] Marko Heikkilä, Matti Pietikäinen, and Cordelia Schmid. Description of interest regions with local binary patterns. *Pattern Recognition*, 2009. doi:10.1016/j.patcog.2008.08.014.

[17] Mitisha Narottambhai Patel and Purvi Tandel. A survey on feature extraction techniques for shape based object recognition. *International Journal of Computer Applications*, 2016. doi:10.5120/ijca2016908782.

[18] Pranjali B. Padol and Anjali A. Yadav. Svm classifier based grape leaf disease detection. *Conference on Advances in Signal Processing (CASP)*, 2016. doi:10.1109/CASP.2016.7746160.

[19] N. Krithika and A. Grace Selvarani. An individual grape leaf disease identification using leaf skeletons and knn classification. *International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 2017. doi:10.1109/ICIIECS.2017.8275951.

[20] Anand K. Hase, Priyanka S. Aher, and Sudeep K. Hase. Detection, categorization and suggestion to cure infected plants of tomato and grapes by using opencv framework for andriod environment. *2nd International Conference for Convergence in Technology (I2CT)*, 2017. doi:10.1109/I2CT.2017.8226270.

[21] Mahantesh C. Elemmi, Shanta Kallur, Nitin Rameshi, Nikhil Aivalli, Prabhat Nayaki, and Adarsh Naik. Vision based classification of different diseases of grape leaves and their severity. *MAT Journals*, 2017.

[22] Harshal Waghmare and Radha Kokare. Detection and classification of diseases of grape plant using opposite colour local binary pattern feature and machine learning for automated decision support system. *3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, 2016. doi:10.1109/SPIN.2016.7566749.

[23] Biswas Sandika, Saunshi Avil, Sarangi Sanat, and Pappula Srinivasu. Random forest based classification of diseases in grapes from images captured in uncontrolled environments. *IEEE 13th International Conference on Signal Processing (ICSP)*, 2016. doi:10.1109/ICSP.2016.7878133.

[24] Pranjali B. Padol and S. D. Sawant. Fusion classification technique used to detect downy and powdery mildew grape leaf diseases. *International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 2016. doi:10.1109/ICGTSPICC.2016.7955315.

[25] Prathamesh K. Kharde and Hemangi H. Hulkarni. Grape leaf disease detection using embedded processor. *International Research Journal of Engineering and Technology (IRJET)*, 2016.

[26] S. S. Sannakki, V. S. Rajpurohit, V. B. Nargund, and P. Kulkarni. Diagnosis and classification of grape leaf diseases using neural networks. *Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 2013. doi:10.1109/ICCCNT.2013.6726616.

[27] Kelly H. Zou, Simon K. Warfield, Aditya Bharatha, Clare M.C. Tempany, Michael R. Kaus, Steven J. Haker, William M. Wells III, Ferenc A. Jolesz, and Ron Kikinis. Statistical validation of image segmentation quality based on a spatial overlap index. *PubMed Central (PMC)*, 2004. doi:10.1016/S1076-6332(03)00671-8.